LLMs: Inference & PEFT

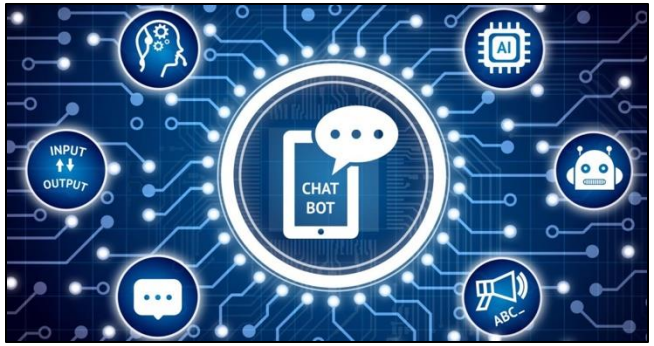
Lecture 14

18-789

Administrative

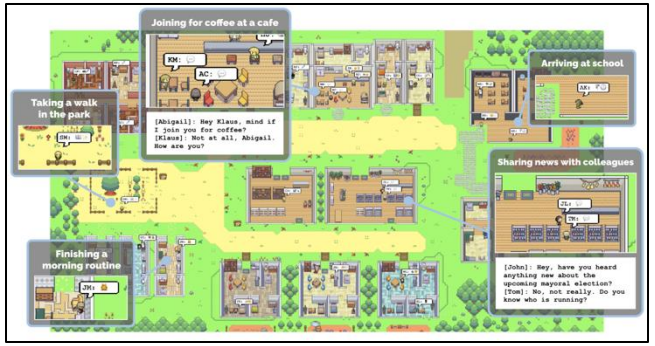
- Today
  - Inference Optimization
  - PEFT

LLMs are Powerful, but Expensive to Deploy



Conversational AI

Content Generation

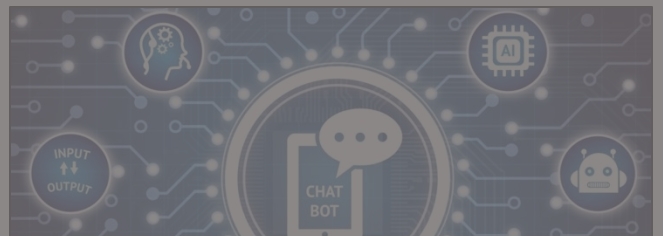


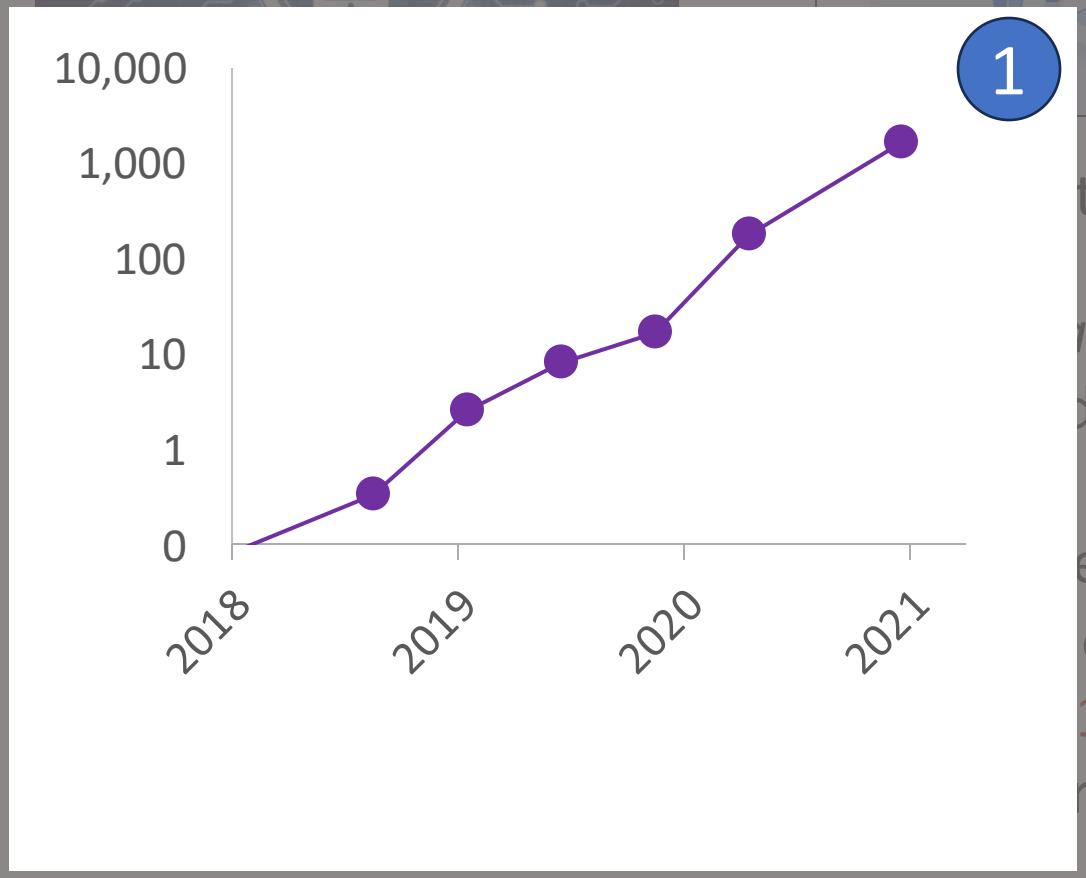
AI Agents

Major Challenges: memory IO (*Pope et al.*) + limited context window

- large mem, e.g. a Llama2-70B model needs
  - 140 GB for parameters,
  - 160 GB for activation (KV cache),
    even with Multi-Group-Attention (8K seqlen + 64 batch size)
- low parallelizability, e.g. generate 100 tokens -> load model, KV cache 100 times
- Perplexity explosion beyond pre-trained windows

LLMs are Powerful, but Expensive to Deploy





t Generation

AI Agents

l.) + limited context window

del needs

e),

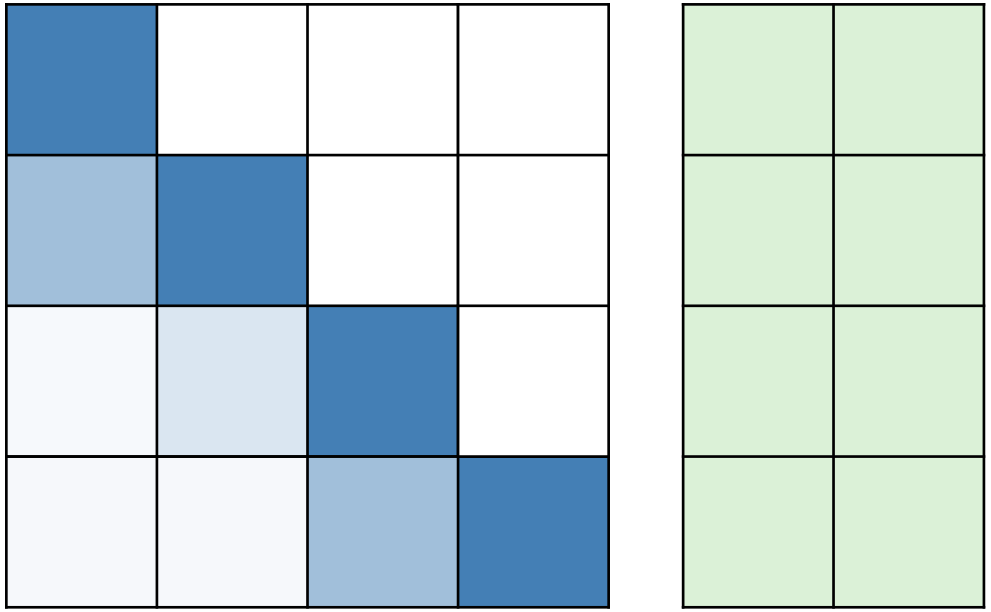
on (8K seqlen + 64 batch size)

100 tokens -> load model, KV cache 100 times

rained windows

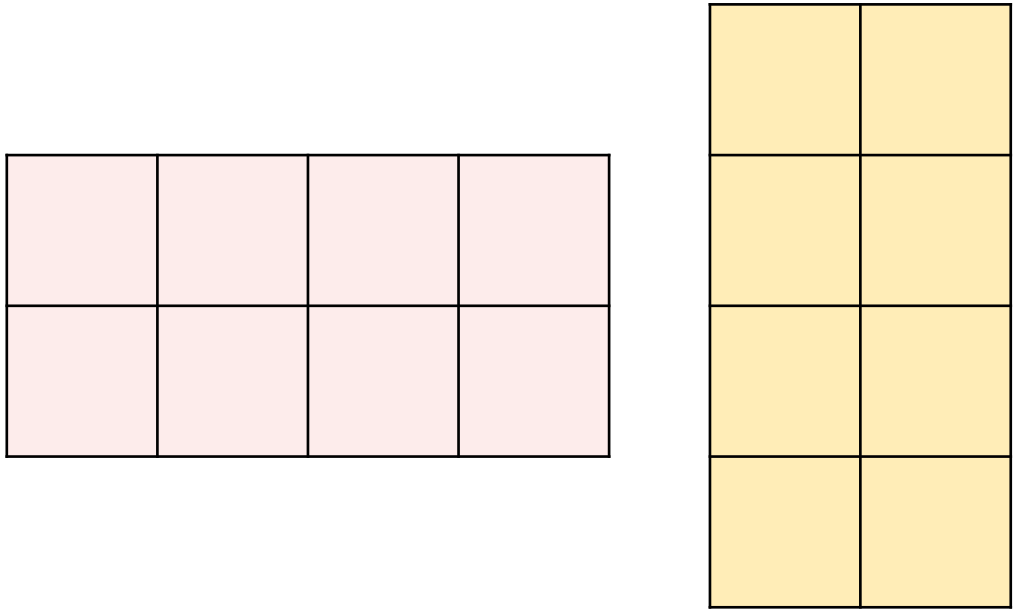
Background: Transformer Architecture

Attention



$\{W_q, W_k, W_v, W_o\ \} \in R^{d\text{x}d}$

MLP



$\{W_1, W_2\} \in R^{d\text{x}4d}$

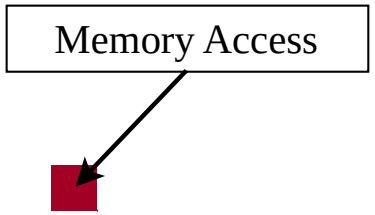
Challenges

The idea sparsity or pruning is not new!

- Long history in ML, statistics, neuroscience, signal processing … *(Lecun et al. 90, Donoho 92, Tibshirani 96, Foldiak et al. 03, Candes et al. 05)*

But hard to speed up sparse LLMs in wall-clock time and maintain quality

- Expensive and infeasible to finetune or retrain
- Difficult to find sparsity that preserves emergent ability of LLMs
- Unstructured sparsity is not hardware-efficient *(Hooker et al. 20)*



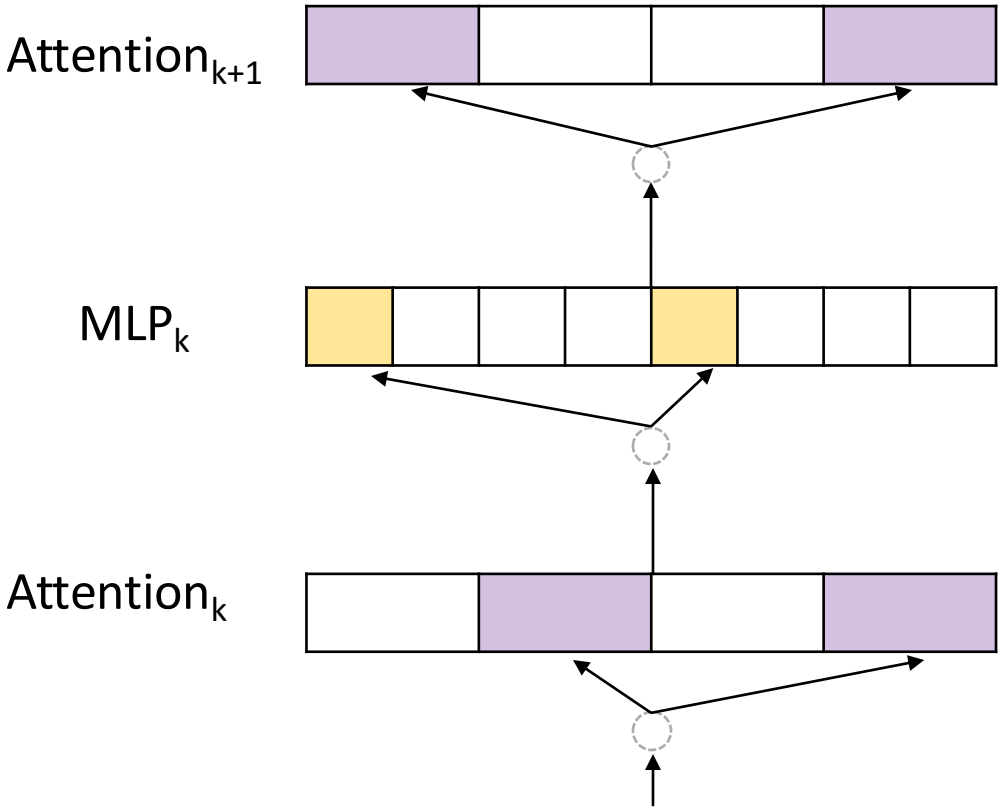
~~Ideal~~ sparsity requires no retraining, maintains quality, and speeds up in wall-clock time.

Contextual

Liu et al. "Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time." (ICML 2023)

Hypothesis: Contextual Sparsity Exists Given Any Input

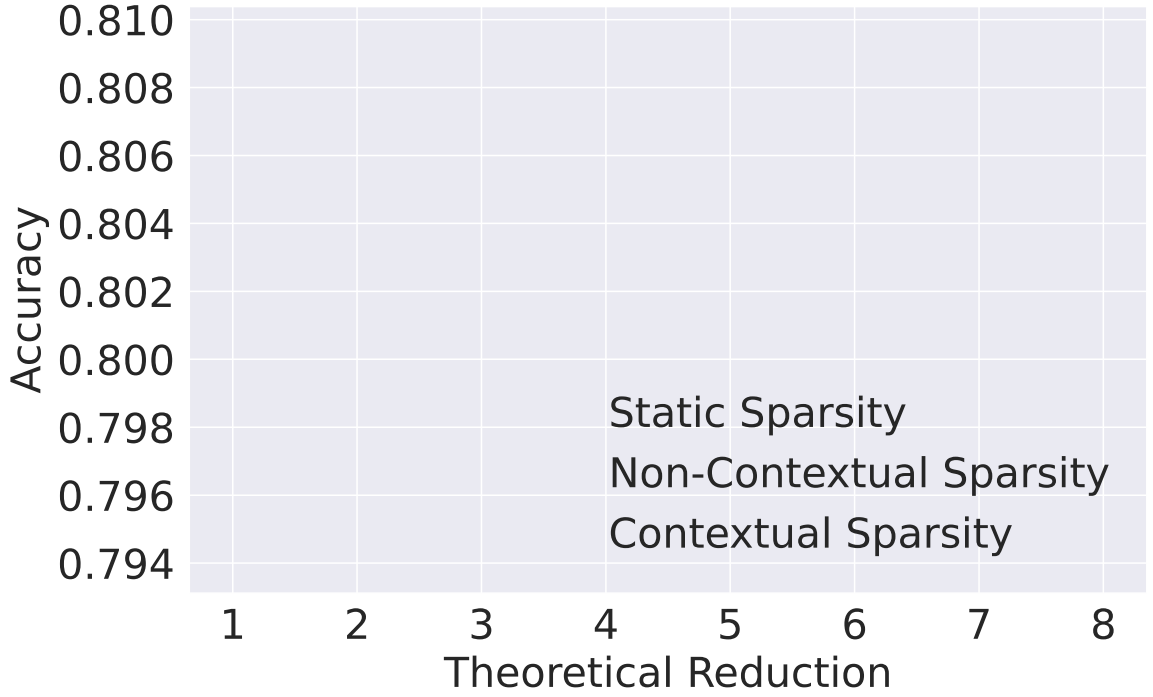
Contextual sparsity: small, input-dependent sets of **attention** heads and **MLP** parameters that lead to (approximately) the same output as the full model for an input.



Inspired by:
connections between LLMs, Hidden Markov Models and Viterbi algorithm (Xie et al.)

Liu et al. "Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time." (ICML 2023)

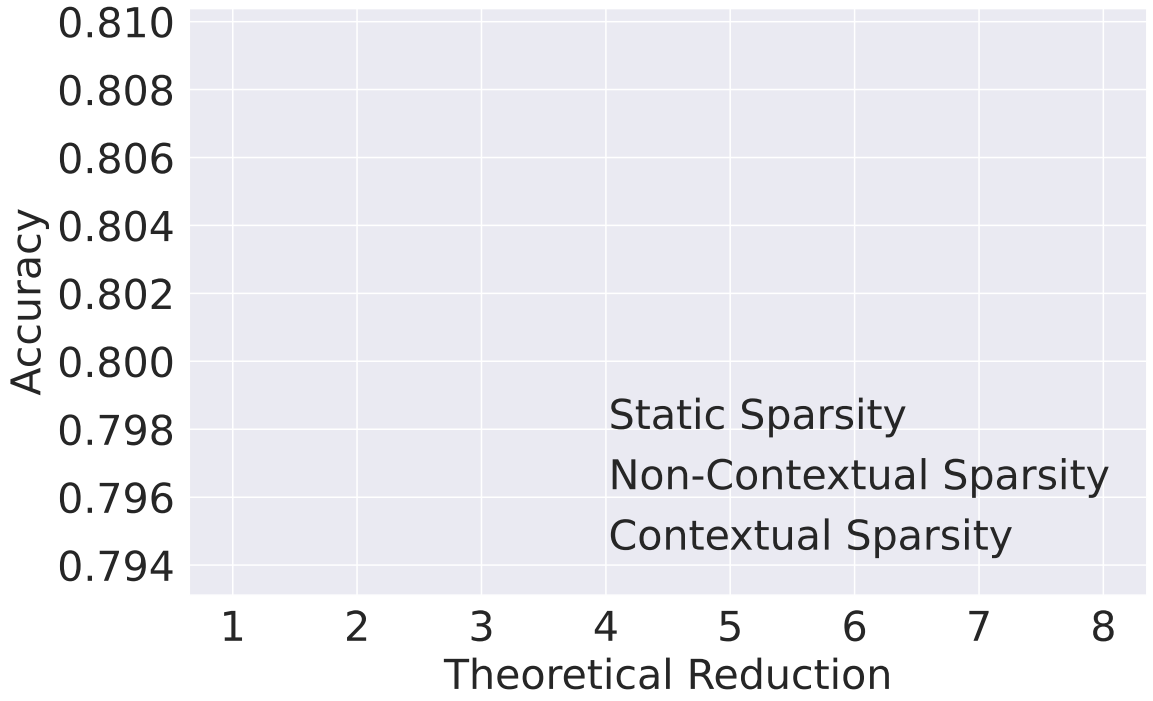
Contextual Sparsity: Existence

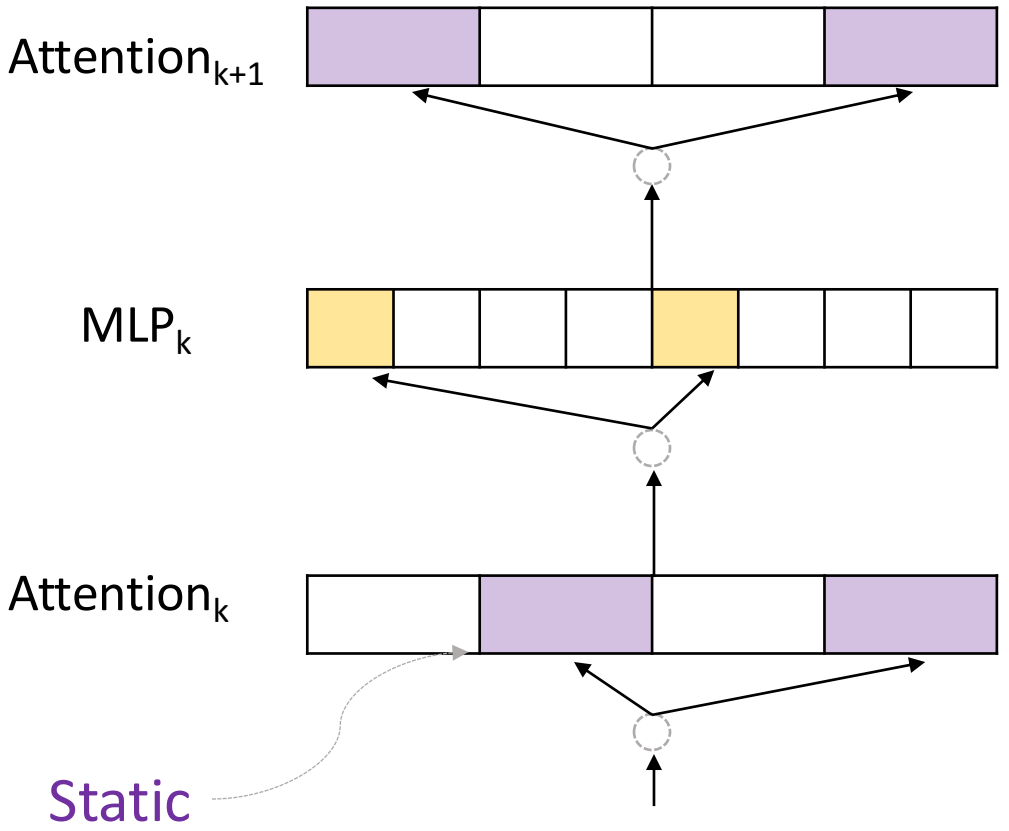


Observation:
keep only high activation in attention/MLP blocks

- 85% structured sparse
  - 80% attention, 95% MLP
- lead to 7× potential parameter reduction for each input
- maintain accuracy

Contextual Sparsity: Existence





LLM Pruning / Sparsity

SparseGPT: Massive Language Models Can be Accurately Pruned in One-Shot

Elias Frantar [1] **Dan Alistarh** [1 2]

Abstract

We show for the first time that large-scale generative pretrained transformer (GPT) family models can be pruned to at least 50% sparsity in *one-shot, without any retraining*, at minimal loss of accuracy. This is achieved via a new pruning method called `SparseGPT`, specifically designed to work efficiently and accurately on massive GPT-family models. We can execute `SparseGPT` on the largest available open-source models, OPT-175B and BLOOM-176B, in under 4.5 hours, and can reach 60% unstructured sparsity with negligible increase in perplexity: remarkably, more than 100 billion weights from these models can be ignored at inference time. `SparseGPT` generalizes to semi-structured (2:4 and 4:8) patterns, and is compatible with weight quantization approaches. The code is available at: https://github.com/IST-DASLab/sparsegpt.

Pruning has a long history (LeCun et al., 1989; Hassibi et al., 1993), and has been applied successfully in the case of vision and smaller-scale language models (Hoefler et al., 2021). Yet, the best-performing pruning methods require *extensive retraining* of the model to recover accuracy. In turn, this is extremely expensive for GPT-scale models. While some accurate *one-shot* pruning methods exist (Hubara et al., 2021a; Frantar et al., 2022b), compressing the model without retraining, unfortunately even they become very expensive when applied to models with billions of parameters. Thus, to date, there is essentially no work on accurate pruning of billion-parameter models.

Overview. In this paper, we propose `SparseGPT`, the first accurate one-shot pruning method which works efficiently at the scale of models with 10-100+ billion parameters. `SparseGPT` works by reducing the pruning problem to a set of extremely large-scale instances of *sparse regression*. It then solves these instances via a new approximate sparse regression solver, which is efficient enough to execute in a few hours on the largest openly-available GPT models (175B parameters), on a single GPU. At the same time, `SparseGPT` is accurate enough to drop negligible accuracy post-pruning,

$$\text{argmin}_{\text{mask } M_\ell, \hat{W}_\ell} \|W_\ell X_\ell - (M_\ell \odot \hat{W}_\ell) X_\ell\|_2^2.$$

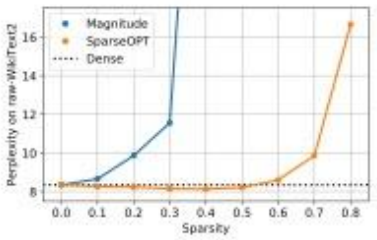


Figure 1. Sparsity-vs-perplexity comparison of `SparseGPT` against magnitude pruning on OPT-175B, when pruning to different *uniform* per-layer sparsities.

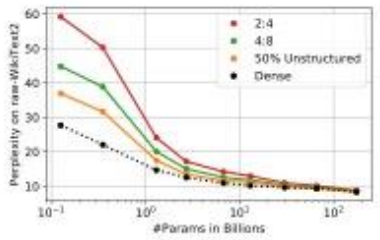


Figure 2. Perplexity vs. model and sparsity type when compressing the entire OPT model family (135M, 350M, ..., 66B, 175B) to different sparsity patterns using `SparseGPT`.

Data-Dependent Static Pruning

A Simple and Effective Pruning Approach for Large Language Models

Mingjie Sun[1*] **Zhuang Liu**[2*] **Anna Bair**[1] **J. Zico Kolter**[1,3]
[1]Carnegie Mellon University [2]Meta AI Research [3]Bosch Center for AI

Abstract

As their size increases, Large Languages Models (LLMs) are natural candidates for network pruning methods: approaches that drop a subset of network weights while striving to preserve performance. Existing methods, however, require either retraining, which is rarely affordable for billion-scale LLMs, or solving a weight reconstruction problem reliant on second-order information, which may also be computationally expensive. In this paper, we introduce a novel, straightforward yet effective pruning method, termed Wanda (Pruning by **W**eights **and a**ctivations), designed to induce sparsity in pretrained LLMs. Motivated by the recent observation of emergent large magnitude features in LLMs, our approach prunes weights with the smallest magnitudes multiplied by the corresponding input activations, on a *per-output* basis. Notably, Wanda requires *no* retraining or weight update, and the pruned LLM can be used *as is*. We conduct a thorough evaluation of our method Wanda on LLaMA and LLaMA-2 across various language benchmarks. Wanda significantly outperforms the established baseline of magnitude pruning and performs competitively against recent method involving intensive weight update. Code is available at https://github.com/locuslab/wanda.

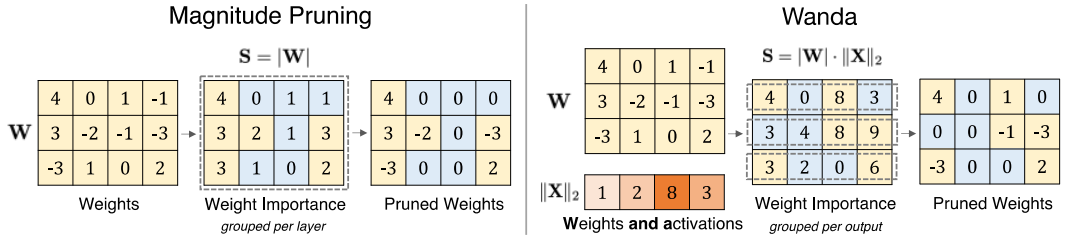
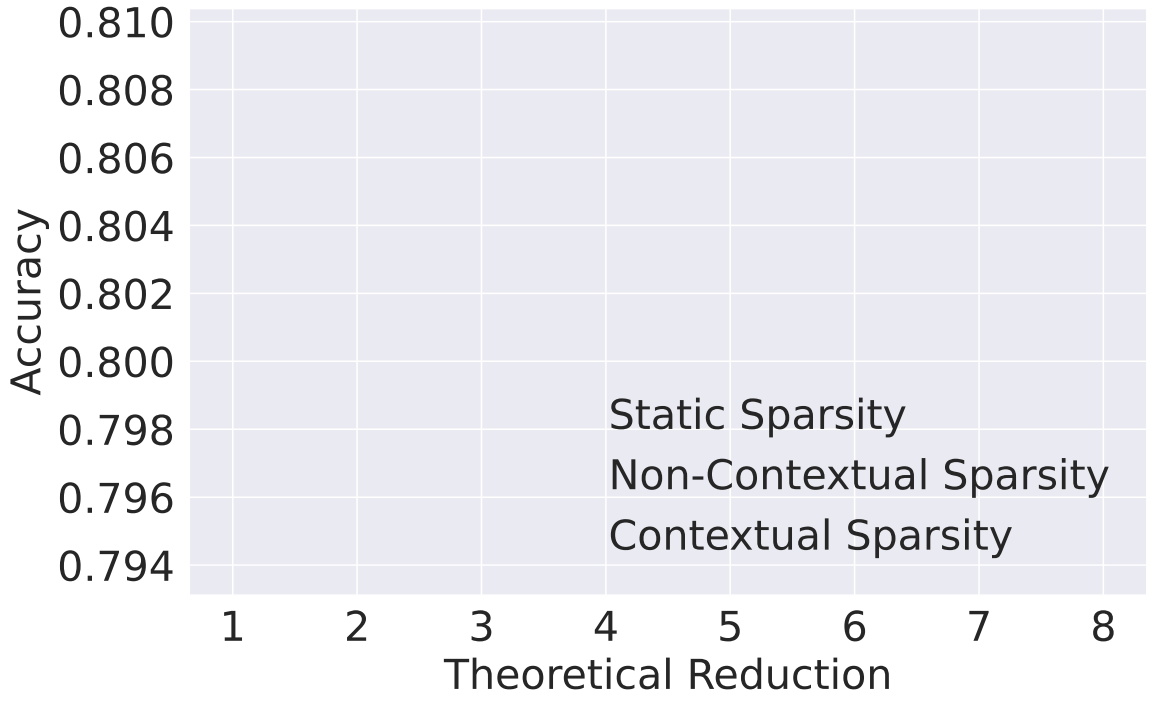
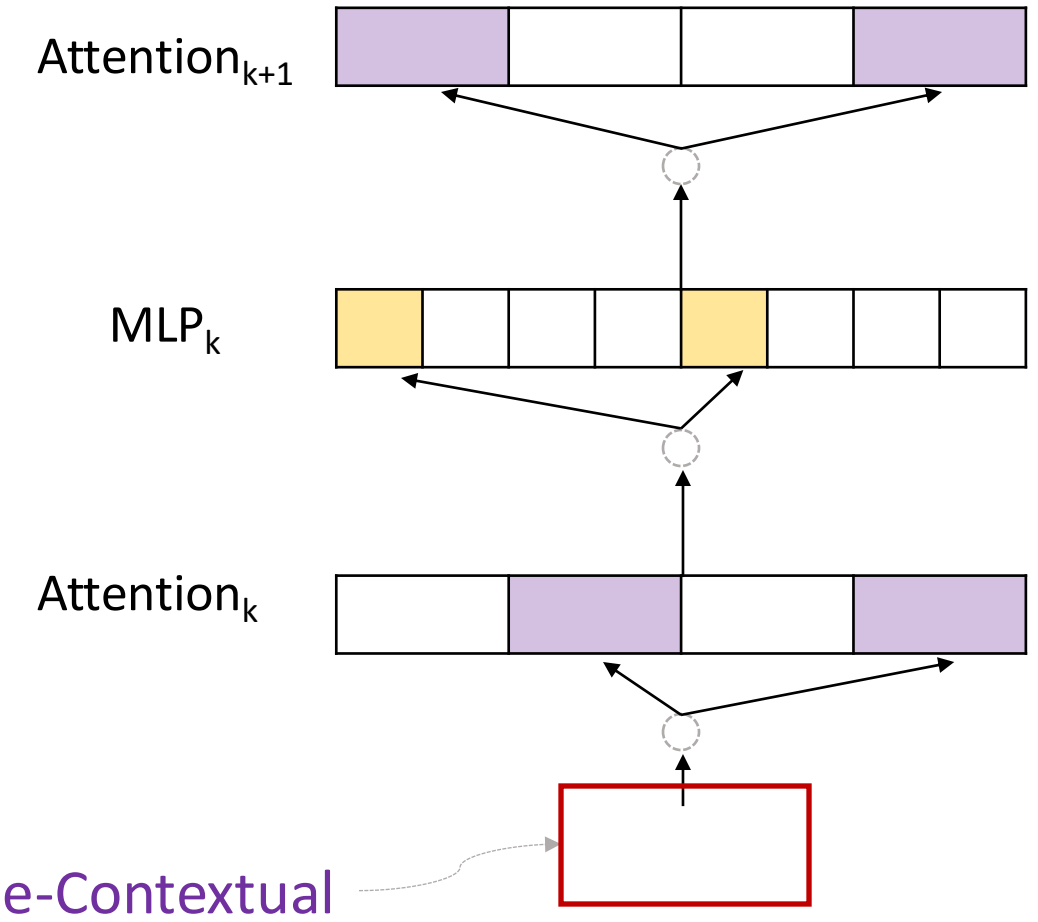


Figure 1: Illustration of our proposed method Wanda (Pruning by **W**eights **and a**ctivations), compared with the magnitude pruning approach. Given a weight matrix $\mathbf{W}$ and input feature activations $\mathbf{X}$, we compute the weight importance as the elementwise product between the weight magnitude and the norm of input activations ($|\mathbf{W}| \cdot \|\mathbf{X}\|_2$). Weight importance scores are compared on a *per-output* basis (within each row in $\mathbf{W}$), rather than globally across the entire matrix.

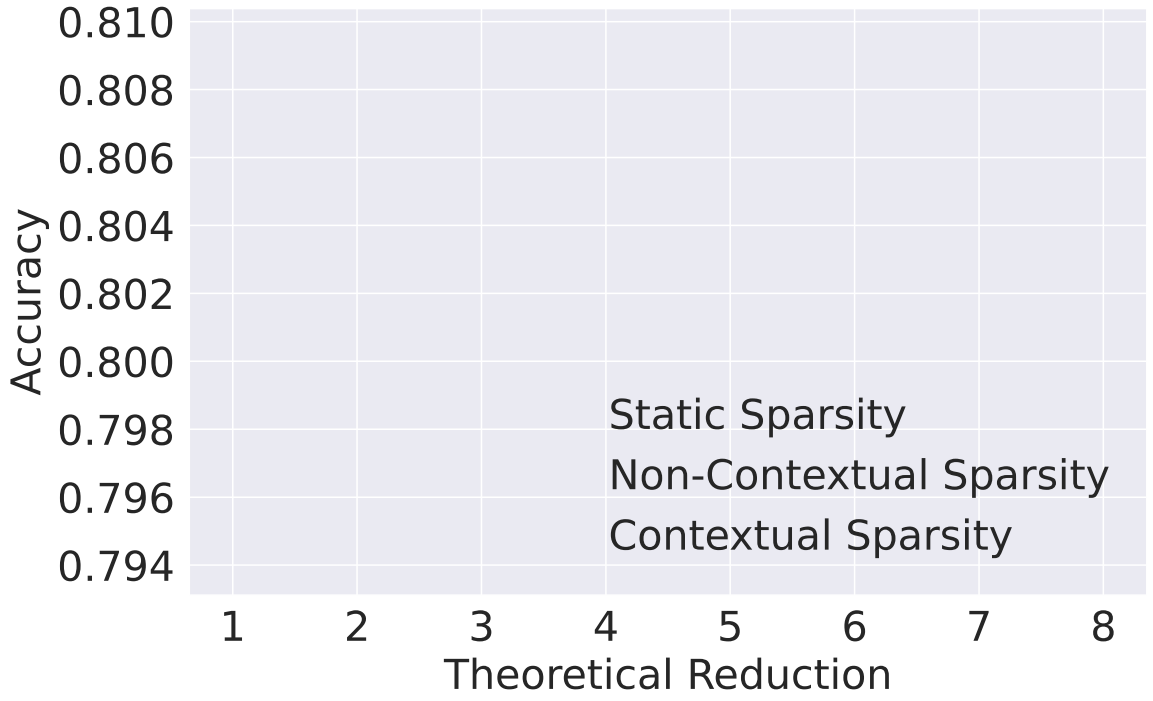
Method	Weight Update	Calibration Data	Pruning Metric $\mathbf{S}_{ij}$	Complexity
Magnitude	✗	✗	$\lvert\mathbf{W}_{ij}\rvert$	$O(1)$
SparseGPT	✓	✓	$\left[\lvert\mathbf{W}\rvert^2/\mathrm{diag}\left[(\mathbf{X}\mathbf{X}^T+\lambda\mathbf{I})^{-1}\right]\right]_{ij}$	$O(d^3_{\texttt{hidden}})$
Wanda	✗	✓	$\lvert\mathbf{W}_{ij}\rvert \cdot \|\mathbf{X}_j\|_2$	$O(d^2_{\texttt{hidden}})$

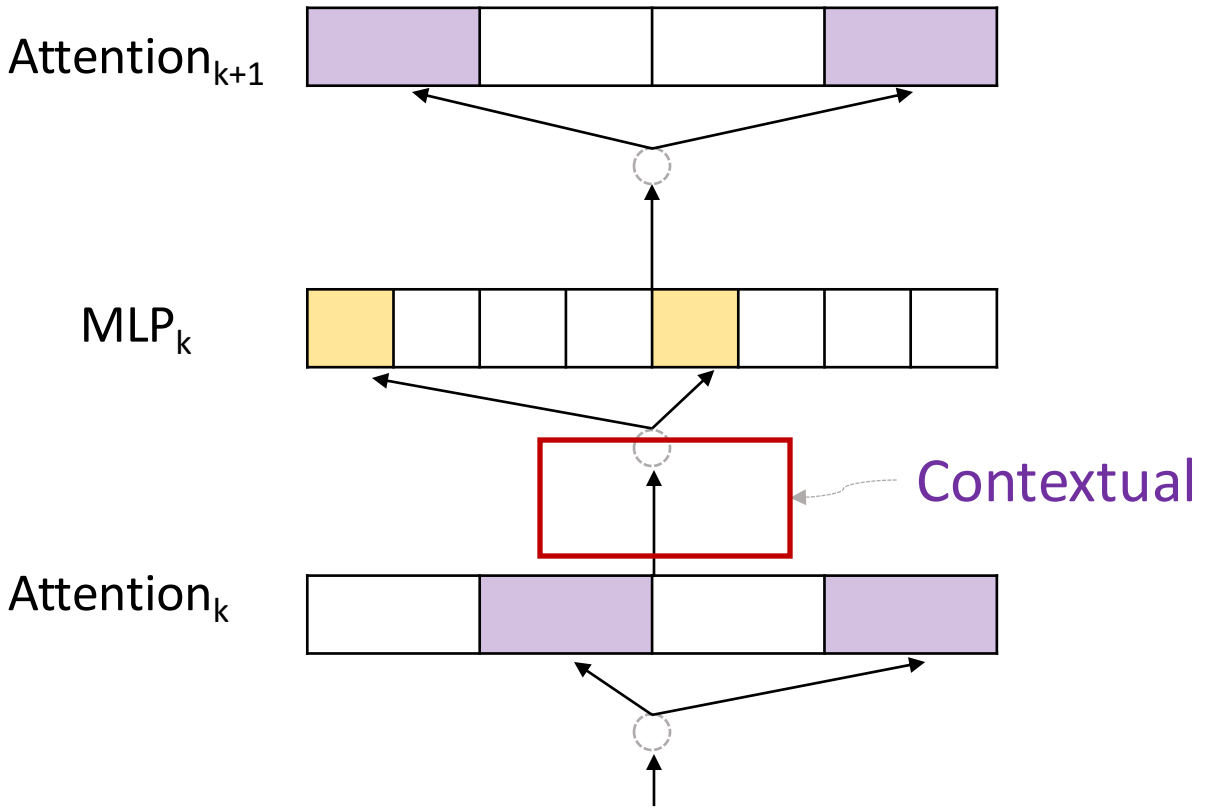
Contextual Sparsity: Existence



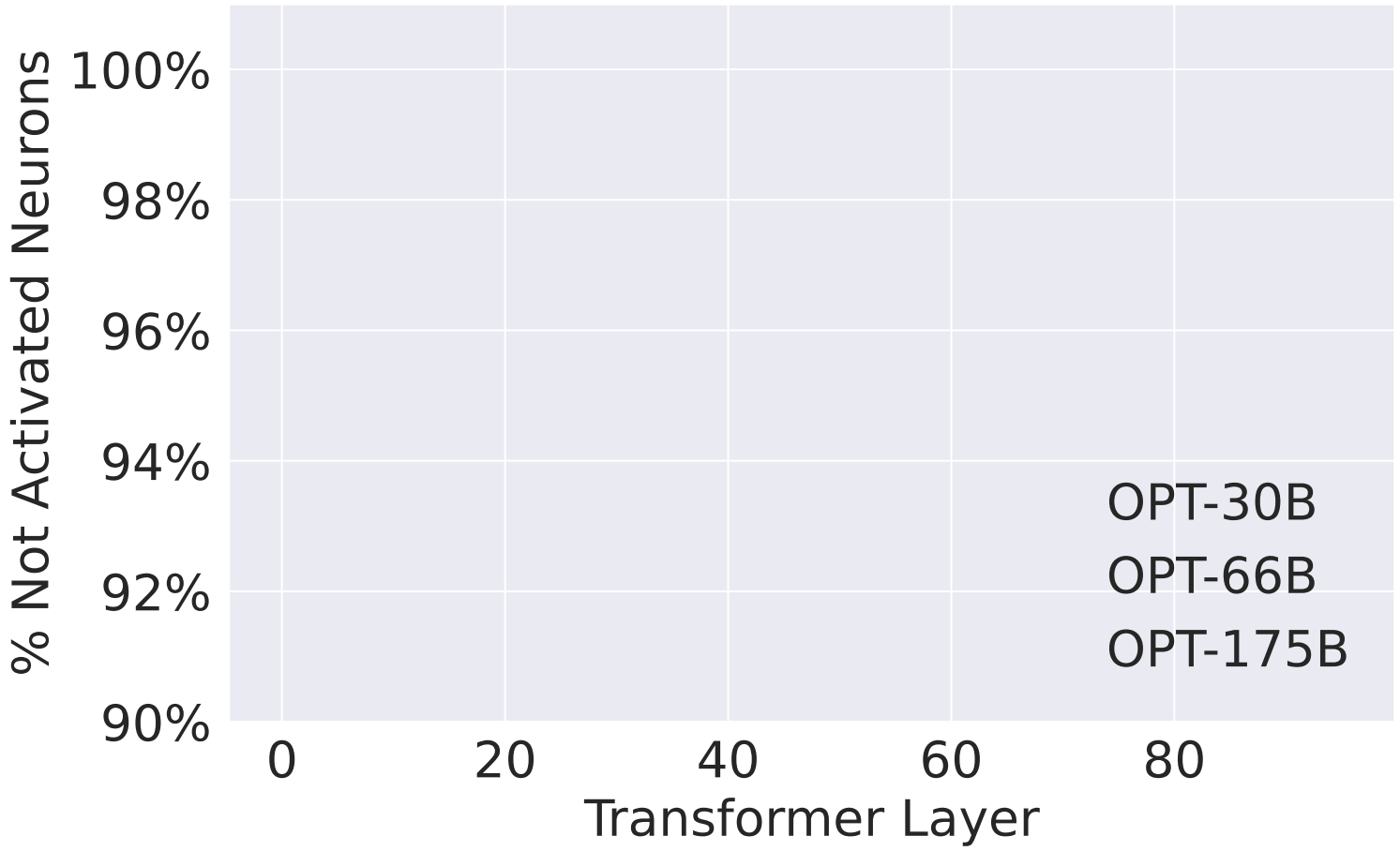


Contextual Sparsity: Existence



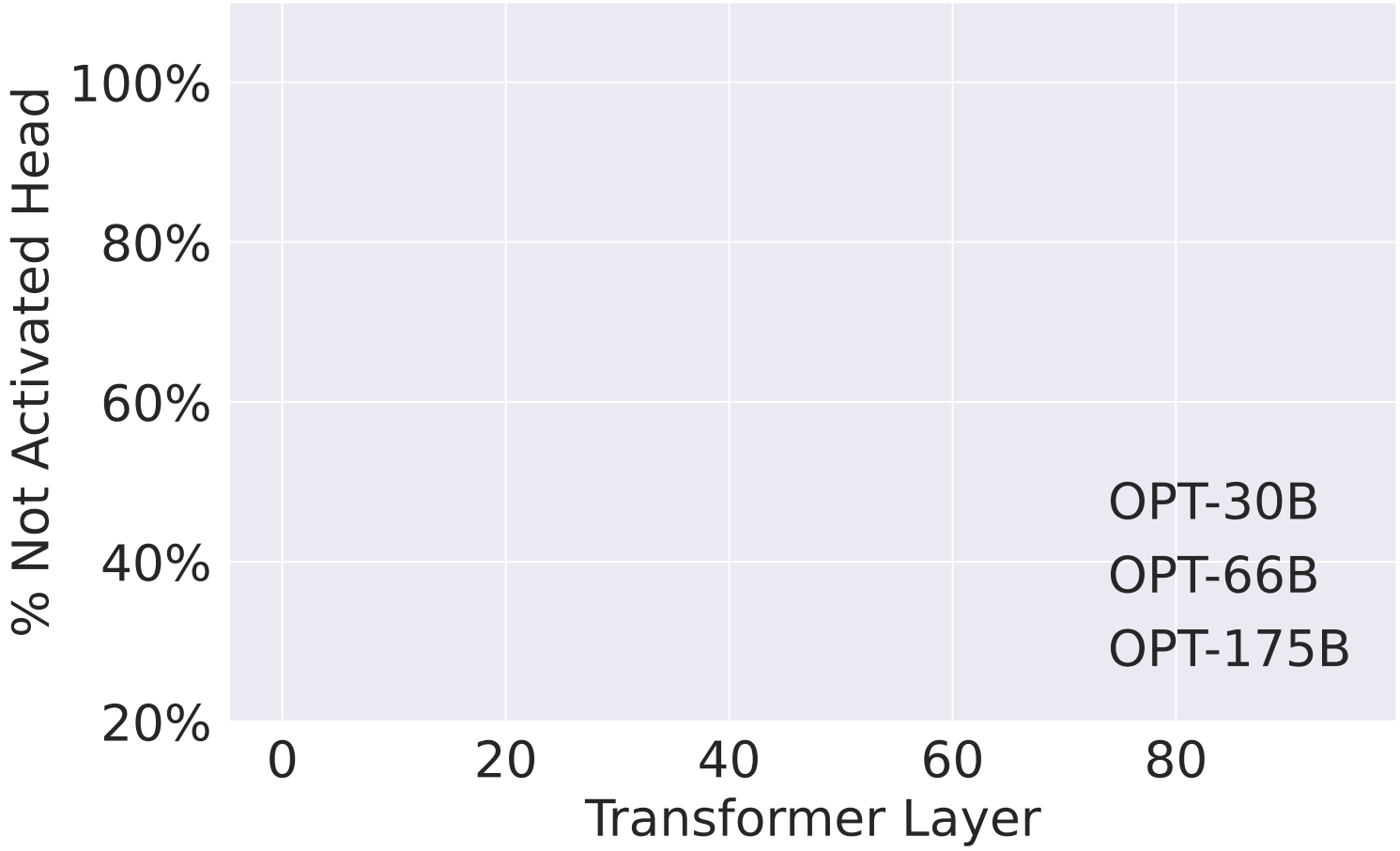


Contextual Sparsity Exists in MLPs

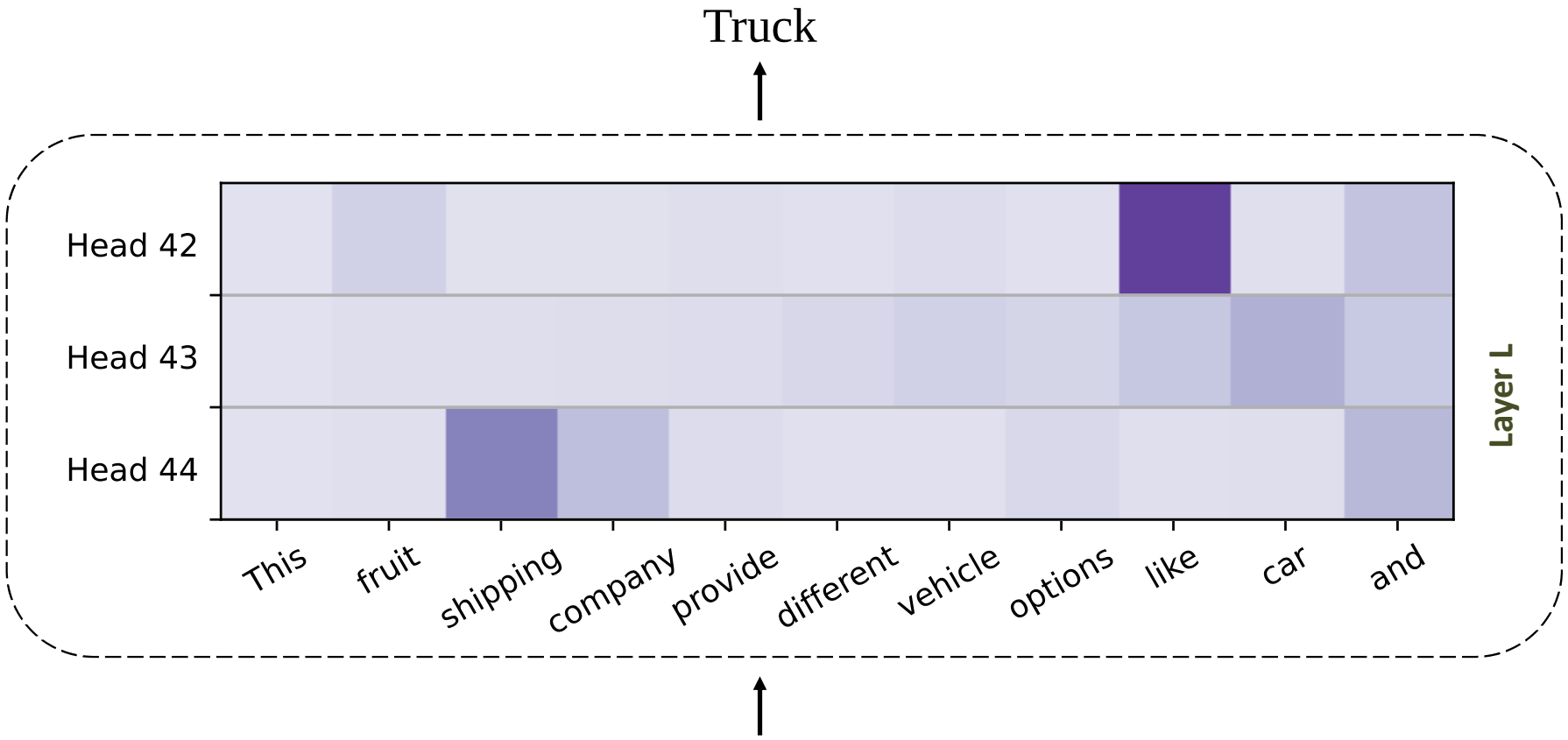


- Due to activation functions, e.g., ReLU, GeLU
- Similar observation in (Li et al.)

Contextual Sparsity Exists in Attention

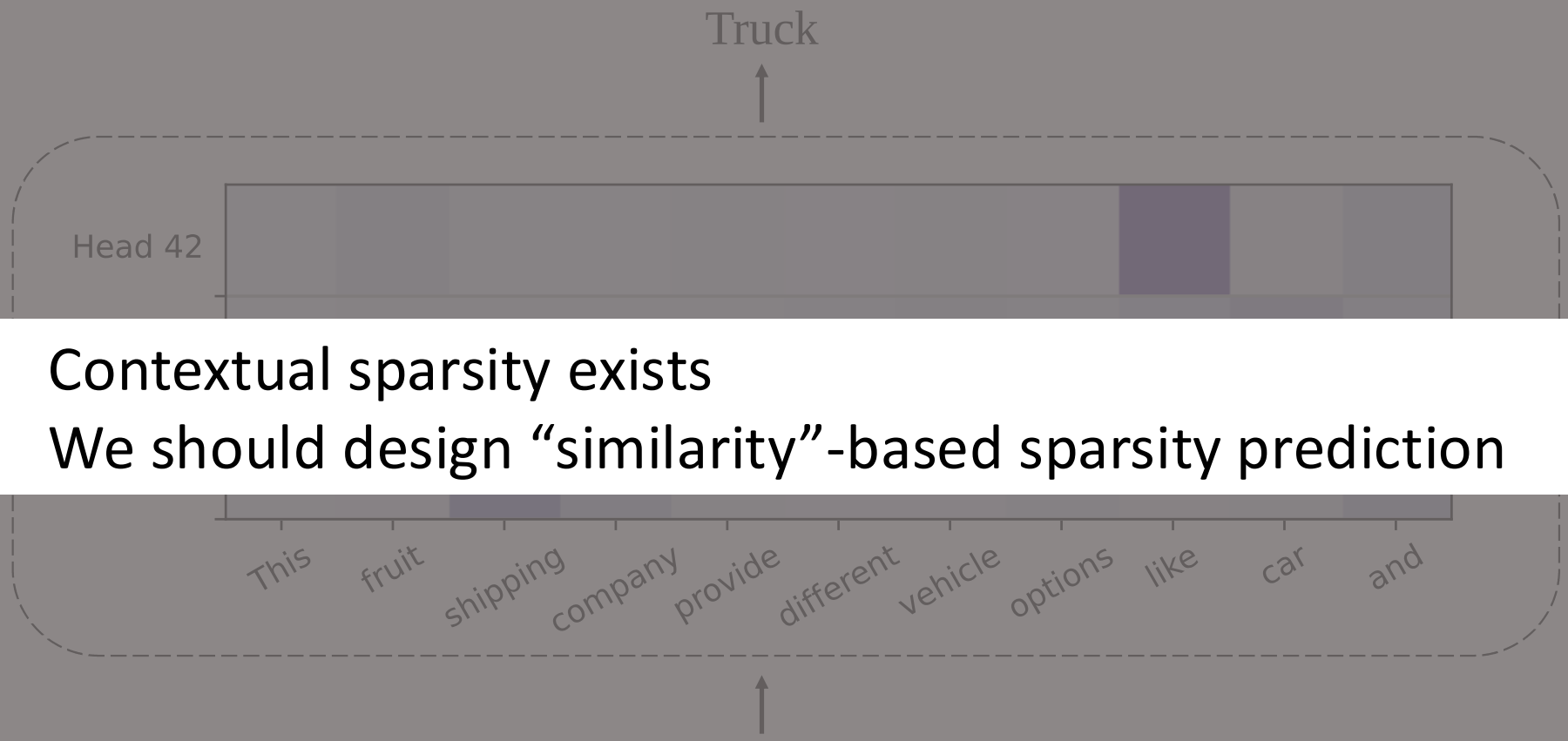


Contextual Sparsity Exists in Attention



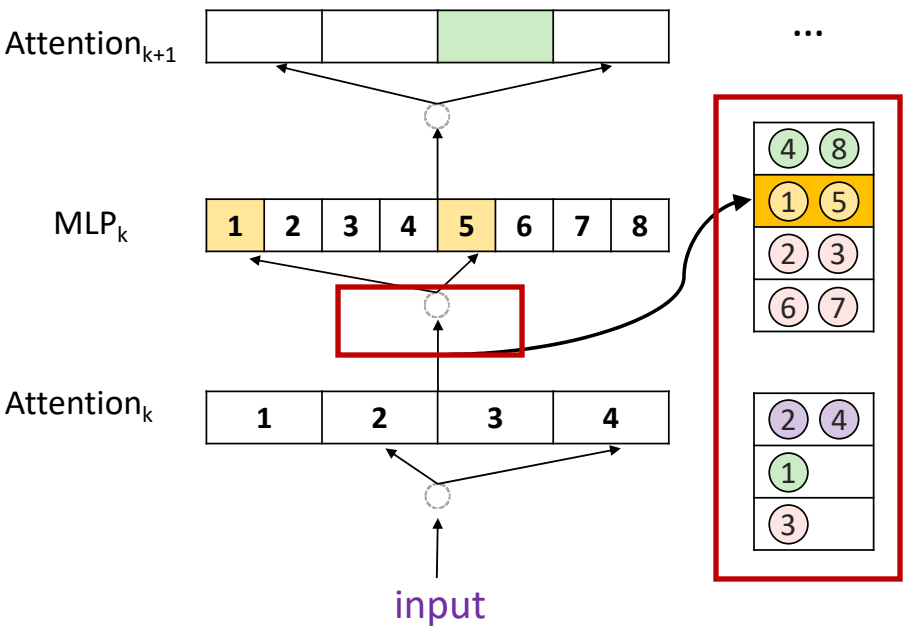
This fruit shipping company provide different vehicle options like car and [MASK]

Contextual Sparsity Exists in Attention



- Contextual sparsity exists
- We should design “similarity”-based sparsity prediction

Contextual Sparsity: Prediction



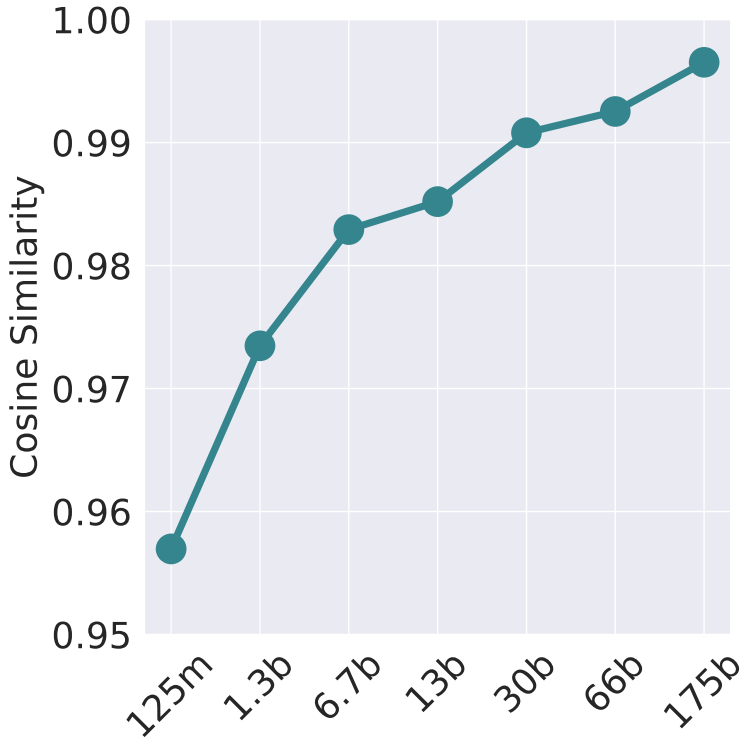
Challenge: how to predict high activation on-the-fly without computing the full attention or MLP?

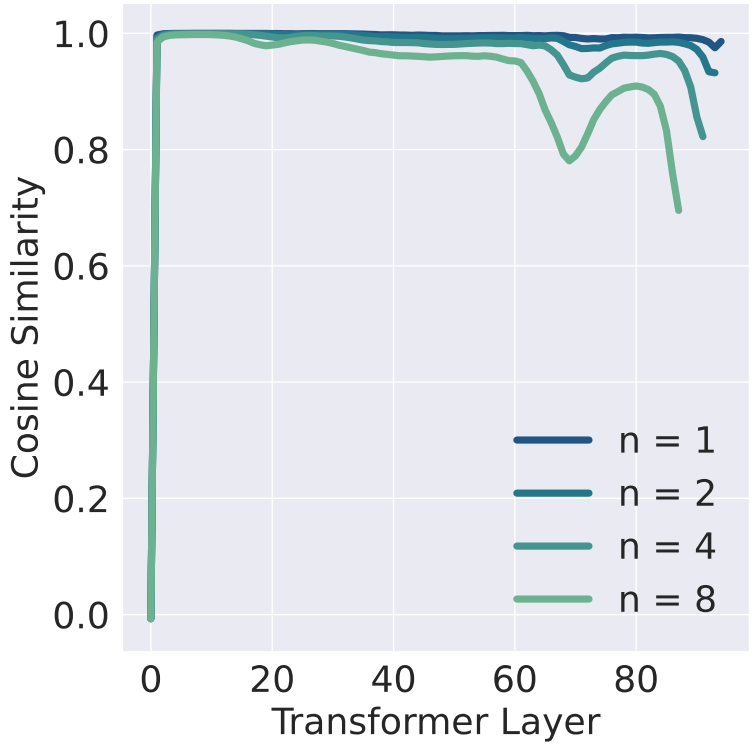
Key idea: design a "similarity"-based prediction

- formulate the prediction problem as near-neighbor search (NNS).
- Data – neurons or attention heads
- Query – input at each layer

NNS algorithms can make prediction based on the similarity between input & parameters.

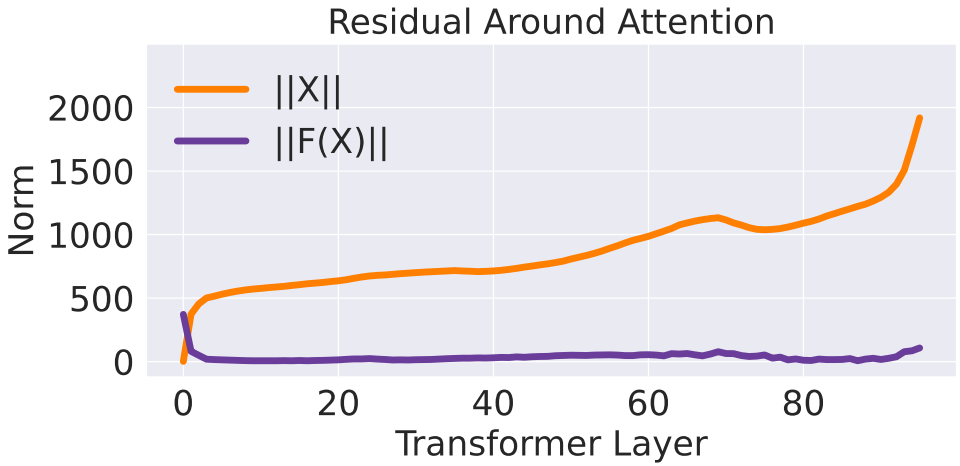
Key Insight: Slowly Changing Embeddings across Layers

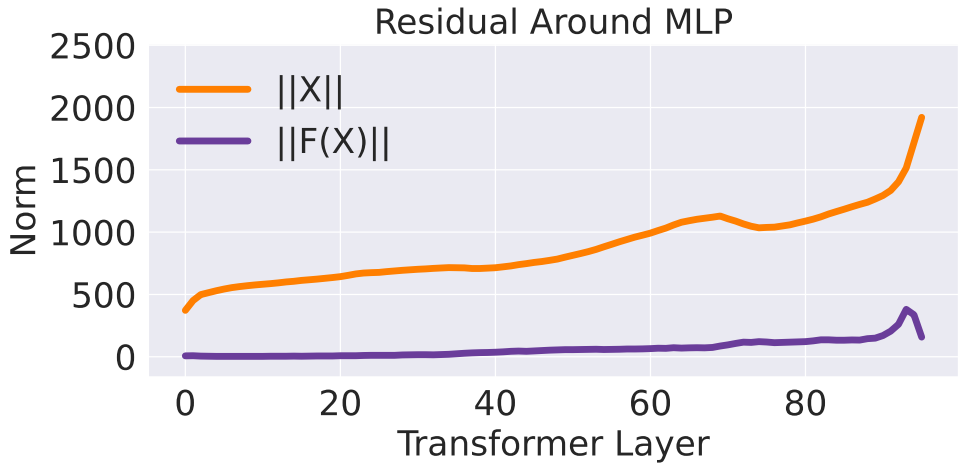




Cosine similarity between representations at consecutive layers is very high.

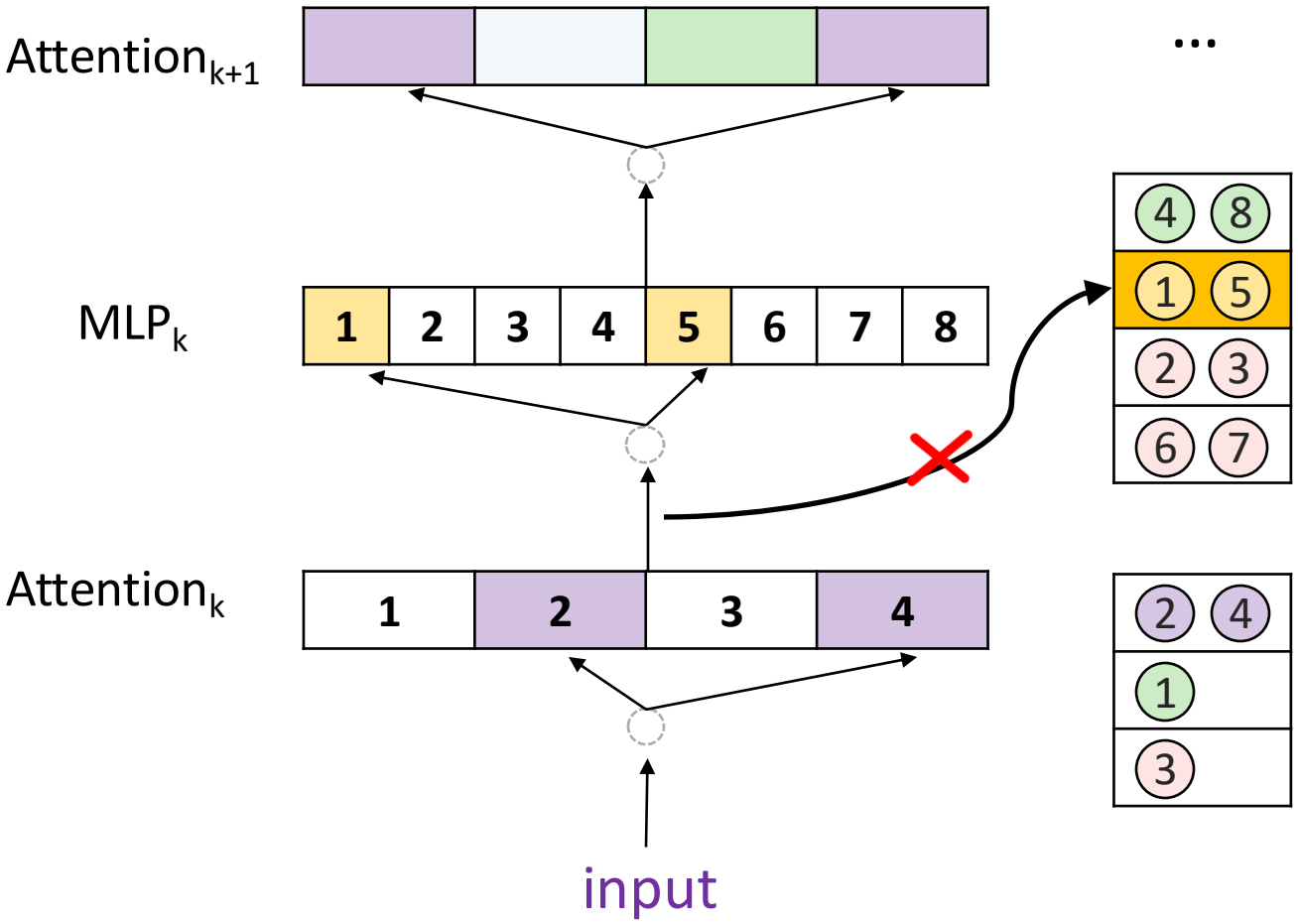
Key Insight: Slowly Changing Embeddings across Layers



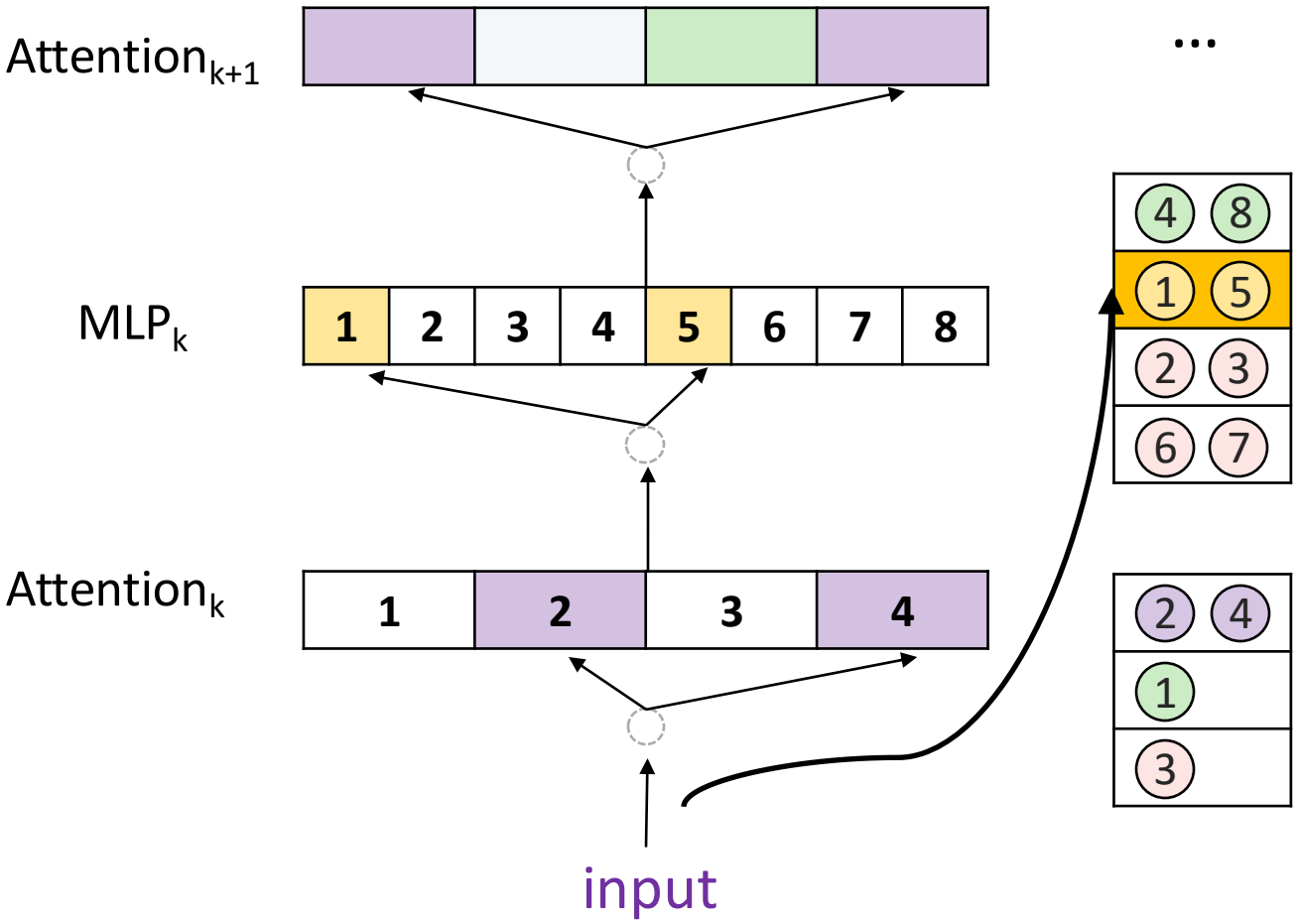


For the residual connection X′ = X +F (X), X's norm dominates.

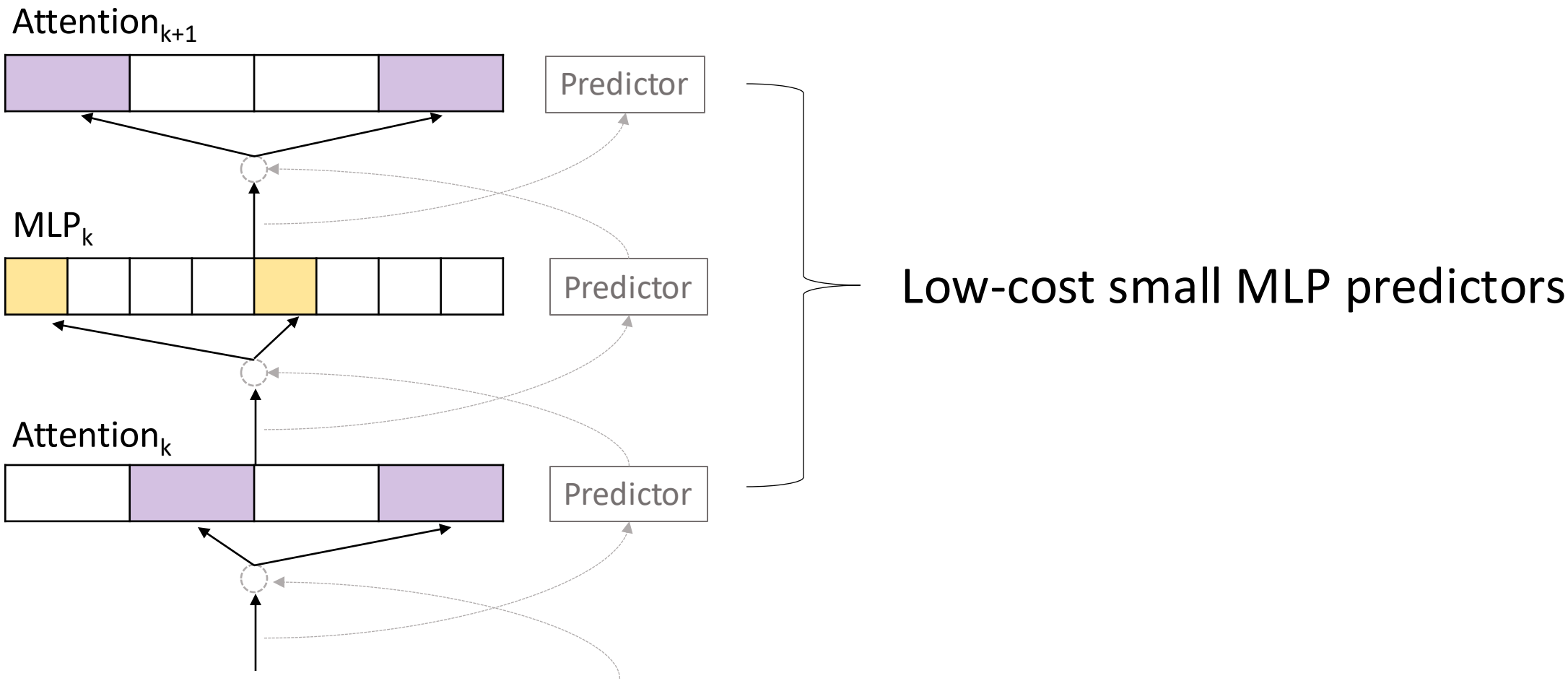
Predict contextual sparsity n-layers ahead



Reduce overhead with Asynchronous Execution



Reduce Overhead with GEMM-based Predictor



Other Contextual Sparsity

CATS: Contextually-Aware Thresholding for Sparsity in Large Language Models

Donghyun Lee[1]
Department of Computer Science
University College London
donghyun.lee.21@ucl.ac.uk

Je-Yong Lee[1]
Mathematical Institute
Oxford University
je-yong.lee@worc.ox.ac.uk

Genghan Zhang, Mo Tiwari, and Azalia Mirhoseini
Department of Computer Science
Stanford University
{zgh23, motiwari, azalia}@stanford.edu

Abstract

The dramatic improvements in Large Language Models (LLMs) come at the cost of increased computational resources for inference. Recent studies ameliorate the computational costs of LLMs by increasing their activation sparsity but suffer from significant performance degradation on downstream tasks. In this work, we introduce a new framework for sparsifying the activations of LLMs and reducing inference costs, dubbed Contextually Aware Thresholding for Sparsity (CATS). CATS is a relatively simple algorithm that is easy to implement and highly effective. At the heart of our framework is a new non-linear activation function. We demonstrate that CATS can be applied to various models, including Mistral-7B and Llama2-7B & 13B, and outperforms existing sparsification techniques across multiple tasks. More precisely, CATS-based models achieve downstream task performance within ∼99% of their base models at 50% activation sparsity, even without fine-tuning. Moreover, with fine-tuning that targets only 1% of the parameters, CATS-based models not only converge faster but also achieve better task performance than competing techniques. Finally, we develop a custom GPU kernel for efficient implementation of CATS that translates the activation sparsity of CATS to real wall-clock time speedups. Our custom kernel implementation of CATS results in a ∼15% improvement in wall-clock inference latency of token generation. We release our code, experiments, and datasets at https://github.com/ScalingIntelligence/CATS.

Prompt-prompted Adaptive Structured Pruning for Efficient LLM Generation

Harry Dong* CMU
Beidi Chen* CMU
Yuejie Chi* CMU

August 13, 2024

Abstract

With the development of transformer-based large language models (LLMs), they have been applied to many fields due to their remarkable utility, but this comes at a considerable computational cost at deployment. Fortunately, some methods such as pruning or constructing a mixture of experts (MoE) aim at exploiting sparsity in transformer feedforward (FF) blocks to gain boosts in speed and reduction in memory requirements. However, these techniques can be very costly and inflexible in practice, as they often require training or are restricted to specific types of architectures. To address this, we introduce GRIFFIN, a novel *training-free* and *calibration-free* method that selects unique FF experts at the sequence level for efficient generation across *a plethora of LLMs with different non-ReLU activation functions*. This is possible due to a critical observation that many trained LLMs naturally produce highly structured FF activation patterns within a sequence, which we call *flocking*. Despite our method's simplicity, we show with 50% of the FF parameters, GRIFFIN maintains the original model's performance with little to no degradation on a variety of classification and generation tasks, all while improving latency (e.g. 1.29× and 1.25× speed-ups in Gemma 7B and Llama 2 13B, respectively, on an NVIDIA L40). Code is available at https://github.com/hdong920/GRIFFIN.

Quantization

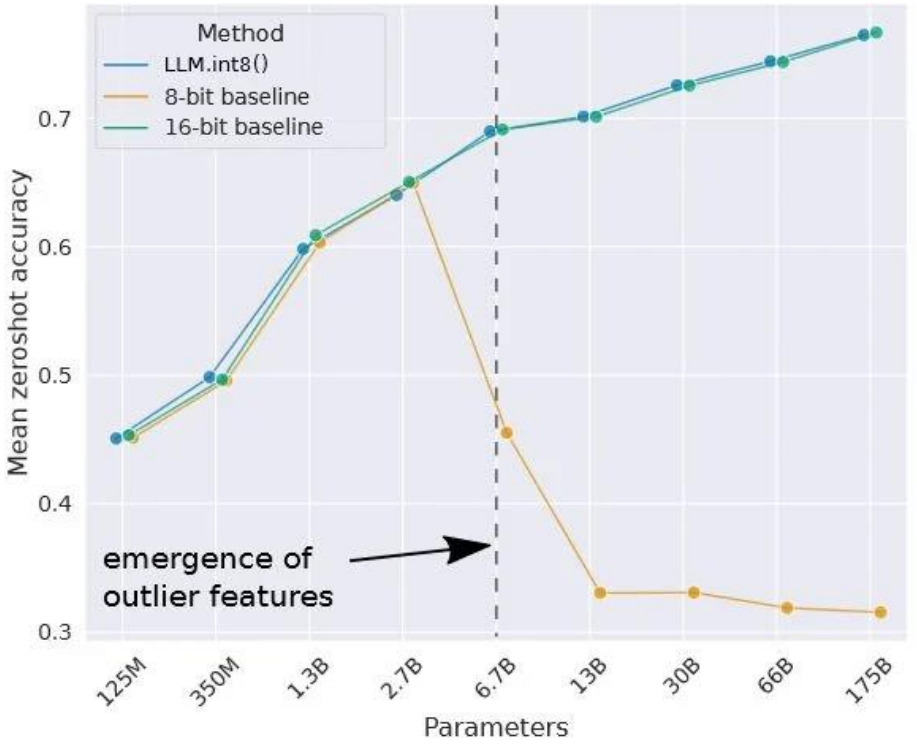
LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale

Tim Dettmers[λ*] **Mike Lewis**[†] **Younes Belkada**[§∓] **Luke Zettlemoyer**[†λ]

University of Washington[λ]
Facebook AI Research[†]
Hugging Face[§]
ENS Paris-Saclay[∓]

Abstract

Large language models have been widely adopted but require significant GPU memory for inference. We develop a procedure for Int8 matrix multiplication for feed-forward and attention projection layers in transformers, which cut the memory needed for inference by half while retaining full precision performance. With our method, a 175B parameter 16/32-bit checkpoint can be loaded, converted to Int8, and used immediately without performance degradation. This is made possible by understanding and working around properties of highly systematic emergent features in transformer language models that dominate attention and transformer predictive performance. To cope with these features, we develop a two-part quantization procedure, **LLM.int8()**. We first use vector-wise quantization with separate normalization constants for each inner product in the matrix multiplication, to quantize most of the features. However, for the emergent outliers, we also include a new mixed-precision decomposition scheme, which isolates the outlier feature dimensions into a 16-bit matrix multiplication while still more than 99.9% of values are multiplied in 8-bit. Using LLM.int8(), we show empirically it is possible to perform inference in LLMs with up to 175B parameters without any performance degradation. This result makes such models much more accessible, for example making it possible to use OPT-175B/BLOOM on a single server with consumer GPUs. We open source our software.



Quantization

SmoothQuant: Accurate and Efficient
Post-Training Quantization for Large Language Models

Guangxuan Xiao[*1] Ji Lin[*1] Mickael Seznec[2] Hao Wu[2] Julien Demouth[2] Song Han[1]
https://github.com/mit-han-lab/smoothquant

Abstract

Large language models (LLMs) show excellent performance but are compute- and memory-intensive. Quantization can reduce memory and accelerate inference. However, existing methods cannot maintain accuracy and hardware efficiency at the same time. We propose SmoothQuant, a training-free, accuracy-preserving, and general-purpose post-training quantization (PTQ) solution to enable 8-bit weight, 8-bit activation (W8A8) quantization for LLMs. Based on the fact that weights are easy to quantize while activations are not, SmoothQuant smooths the activation outliers by offline *migrating* the quantization difficulty from activations to weights with a mathematically equivalent transformation. SmoothQuant enables an INT8 quantization of *both* weights and activations for all the matrix multiplications in LLMs, including OPT, BLOOM, GLM, MT-NLG, Llama-1/2, Falcon, Mistral, and Mixtral models. We demonstrate up to 1.56× speedup and 2× memory reduction for LLMs with negligible loss in accuracy. SmoothQuant enables serving 530B LLM within a single node. Our work offers a turn-key solution that reduces hardware costs and democratizes LLMs.

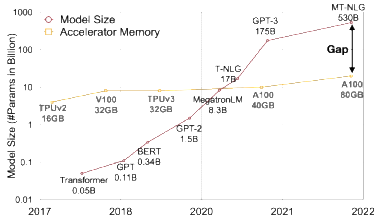
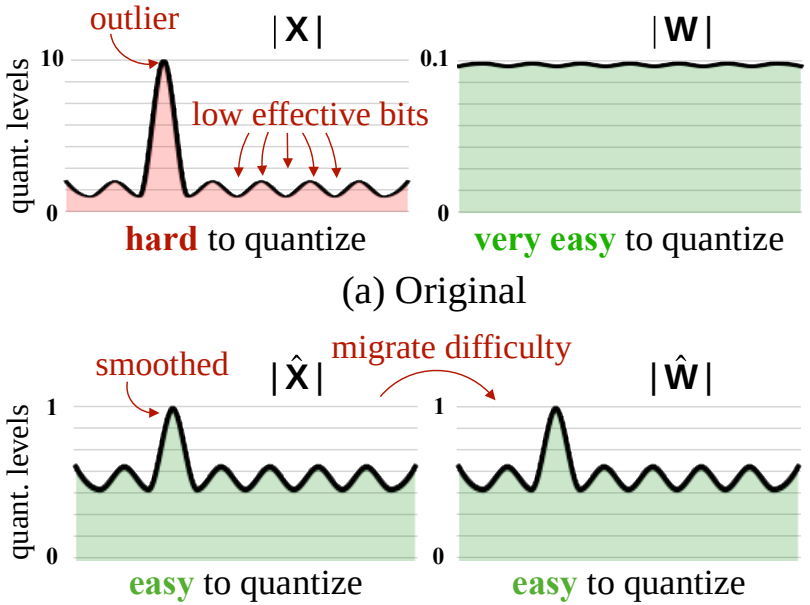


Figure 1: The model size of large language models is developing at a faster pace than the GPU memory in recent years, leading to a big gap between the supply and demand for memory. Quantization and model compression techniques can help bridge the gap.

store and run in FP16, requiring 8→48GB A6000 GPUs or 5→80GB A100 GPUs just for inference. Due to the huge computation and communication overhead, the inference latency may also be unacceptable to real-world applications. *Quantization* is a promising way to reduce the cost of LLMs (Dettmers et al., 2022; Yao et al., 2022). By quantizing the *weights and activations* with low-bit integers, we can reduce GPU memory requirements, in size and bandwidth, and accelerate compute-intensive operations (i.e., GEMM in linear layers, BMM in attention). For instance, INT8



Quantization

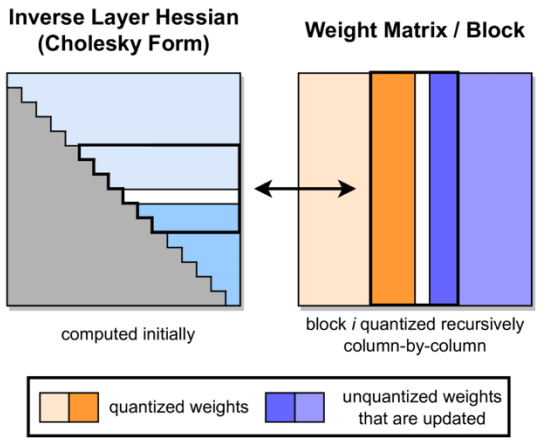
Published as a conference paper at ICLR 2023

GPTQ: ACCURATE POST-TRAINING QUANTIZATION FOR GENERATIVE PRE-TRAINED TRANSFORMERS

Elias Frantar* IST Austria
Saleh Ashkboos ETH Zurich
Torsten Hoefler ETH Zurich
Dan Alistarh IST Austria & NeuralMagic

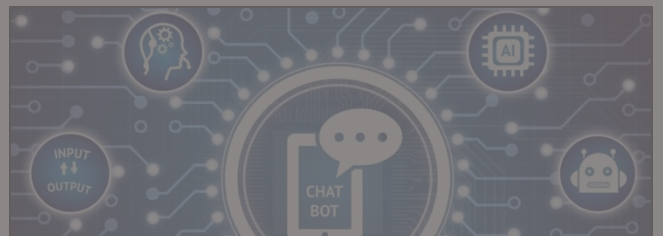
ABSTRACT

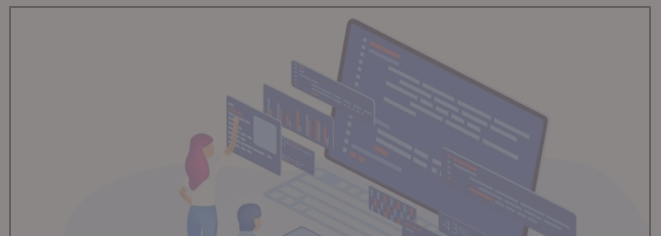
Generative Pre-trained Transformer models, known as GPT or OPT, set themselves apart through breakthrough performance across complex language modelling tasks, but also by their extremely high computational and storage costs. Specifically, due to their massive size, even inference for large, highly-accurate GPT models may require multiple performant GPUs, which limits the usability of such models. While there is emerging work on relieving this pressure via model compression, the applicability and performance of existing compression techniques is limited by the scale and complexity of GPT models. In this paper, we address this challenge, and propose GPTQ, a new one-shot weight quantization method based on approximate second-order information, that is both highly-accurate and highly-efficient. Specifically, GPTQ can quantize GPT models with 175 billion parameters in approximately four GPU hours, reducing the bitwidth down to 3 or 4 bits per weight, with negligible accuracy degradation relative to the uncompressed baseline. Our method more than doubles the compression gains relative to previously-proposed one-shot quantization methods, preserving accuracy, allowing us for the first time to execute an 175 billion-parameter model inside a single GPU for generative inference. Moreover, we also show that our method can still provide reasonable accuracy in the *extreme quantization* regime, in which weights are quantized to 2-bit or even *ternary* quantization levels. We show experimentally that these improvements can be leveraged for end-to-end inference speedups over FP16, of around 3.25x when using high-end GPUs (NVIDIA A100) and 4.5x when using more cost-effective ones (NVIDIA A6000). The implementation is available at `https://github.com/IST-DASLab/gptq`.



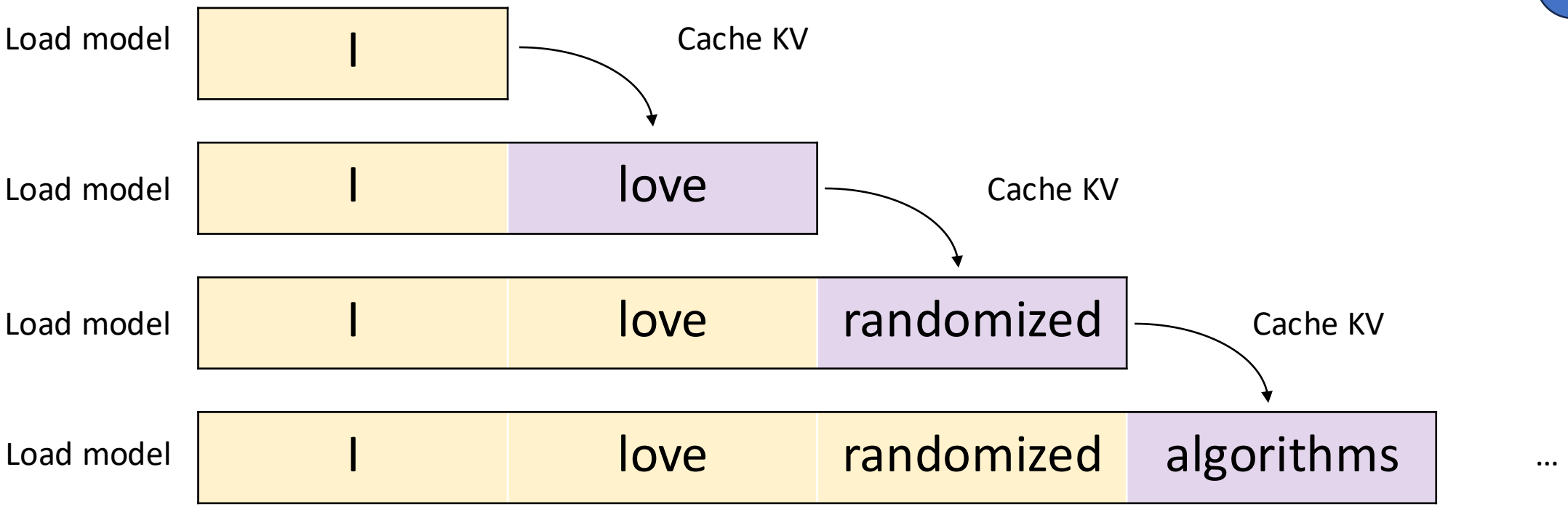
$$\text{argmin}_{\widehat{\mathbf{W}}} ||\mathbf{WX} - \widehat{\mathbf{W}}\mathbf{X}||_2^2.$$

LLMs are Powerful, but Expensive to Deploy



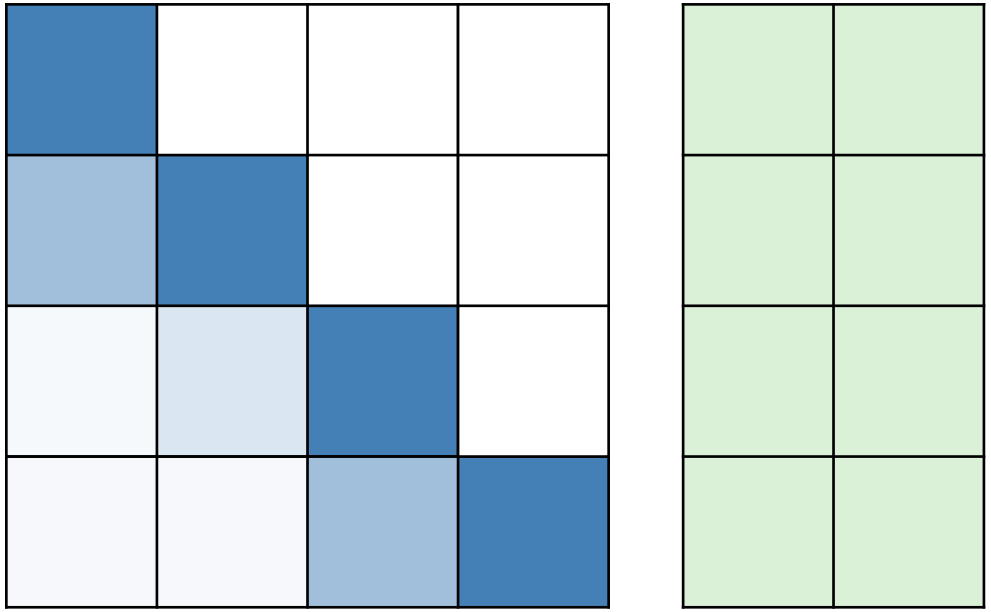


2



Background: Transformer Architecture

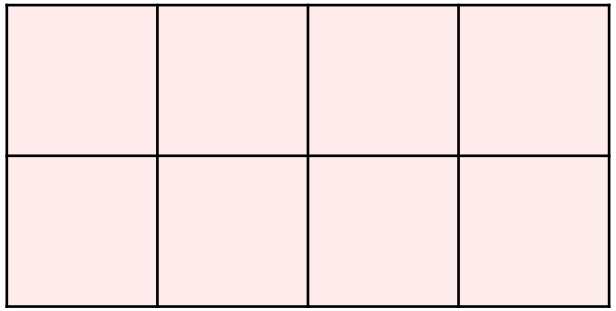
Attention



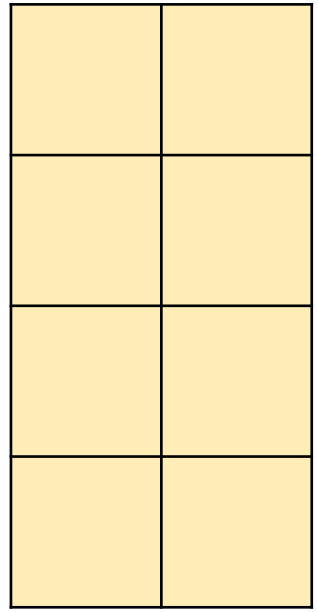
$A = \text{softmax}(Q K^T)$

V

MLP

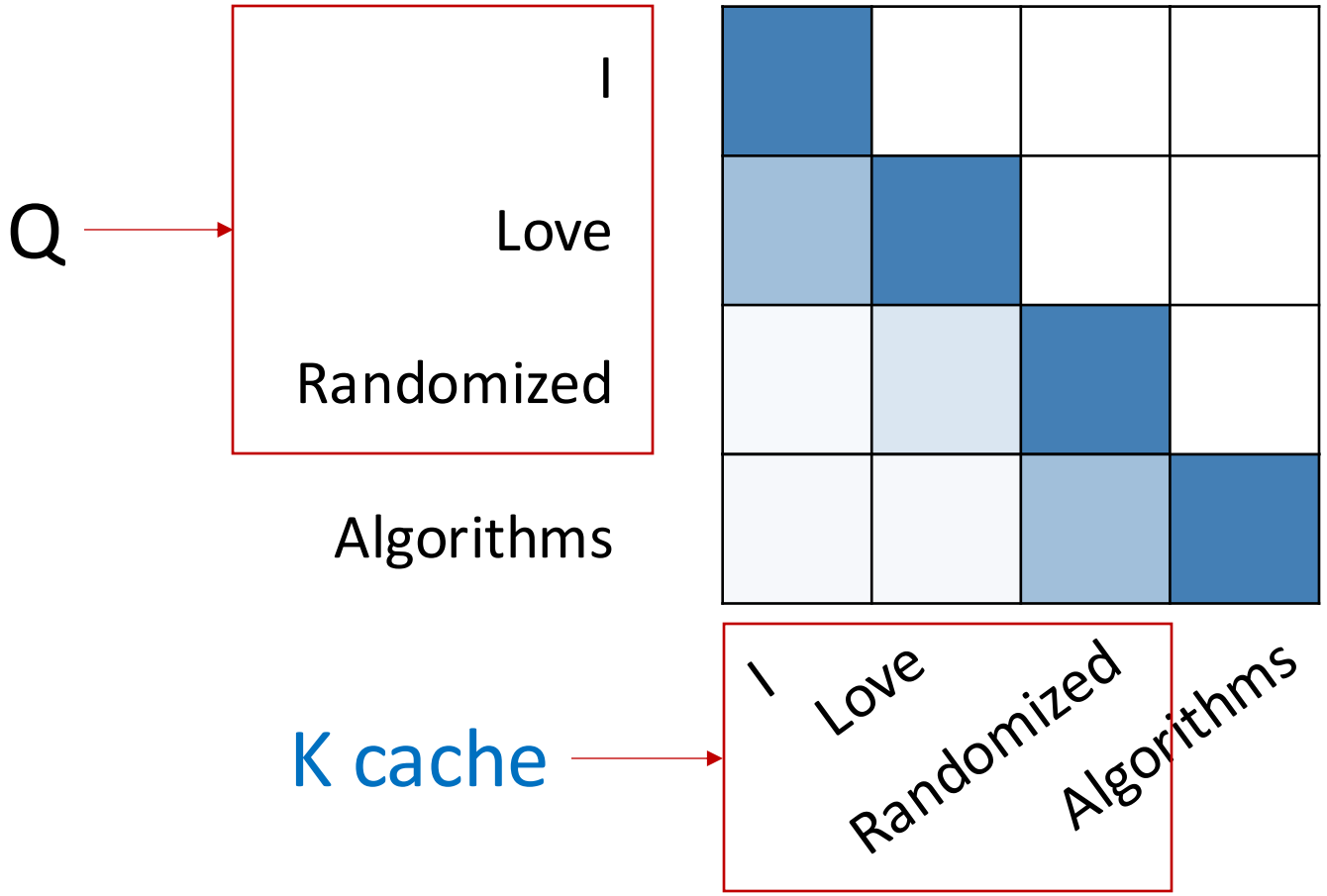


W_1



W_2

KV Cache Bottleneck



KV states for context or previously generated tokens will be cached to avoid re-computation.

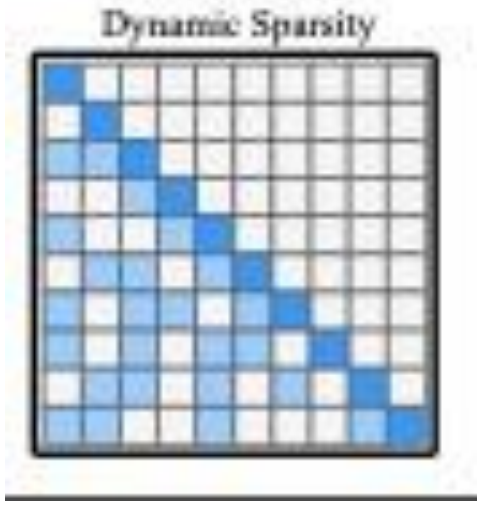
KV cache size scales linearly with sequence length and batch size.

Existing Approaches and Challenges

Naturally, we can limit the cache size like the SW/HW caches. Attention approximation has been widely studied in training long sequences!

But hard to adapt to generation:

- Reduce quadratic attention but not KV cache size
  - e.g., FlashAttention, Reformer

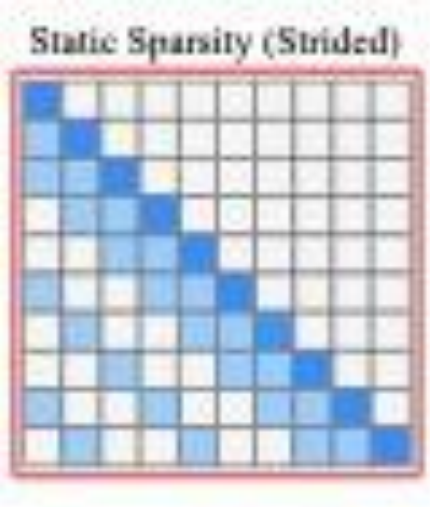


Existing Approaches and Challenges

Naturally, we can limit the cache size like the SW/HW caches. Attention approximation has been widely studied in training long sequences!

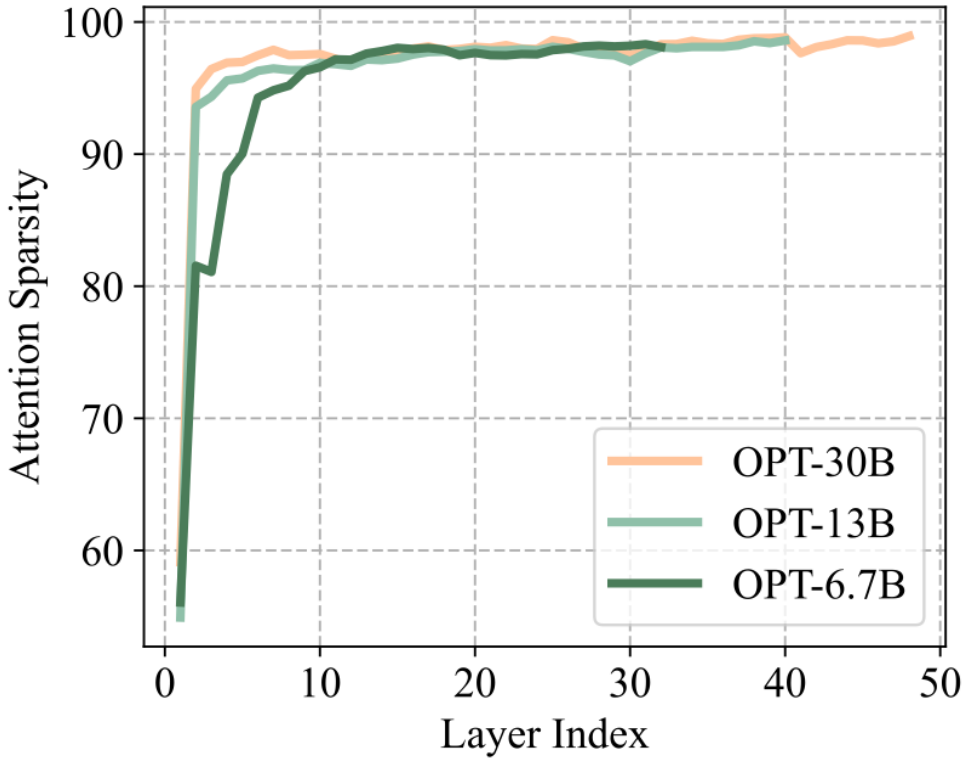
But hard to adapt to generation:

- Reduce quadratic attention but not KV cache size
  - e.g., FlashAttention, Reformer
- Result high cache miss rates and degrade accuracy
  - e.g., Sparse Transformer
- Expensive eviction policy
  - e.g., Gisting Tokens



An ideal cache has a small cache size, a low miss rate, and a low-cost eviction policy.

Sparsity for Smaller Cache Size



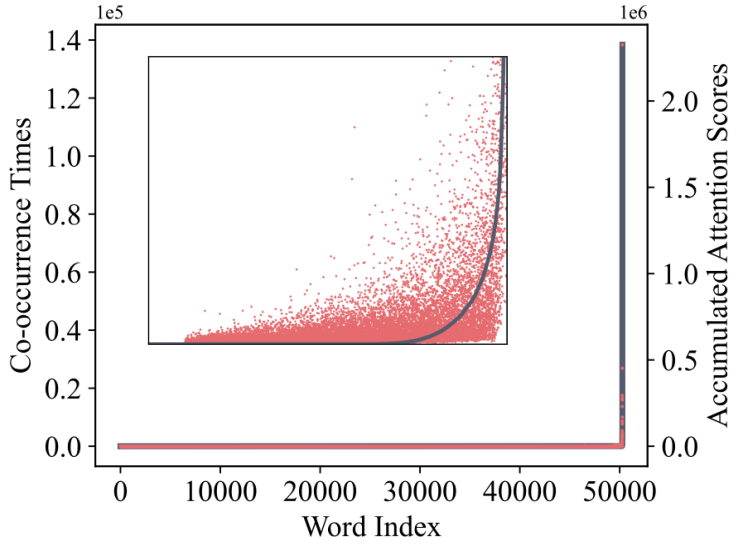
Observation: although densely trained, LLMs

- attention score matrices are highly sparse, with a sparsity over 95% in almost all layers
- leads to 20× potential KV cache reduction
- maintains same accuracy

Attention sparsity widely exists in pre-trained models, e.g. OPT /LLaMA /Bloom/GPT.

Heavy-Hitters for Low Miss Rate

Challenge: how to evict tokens? Once evicted, future tokens can no longer attend to it

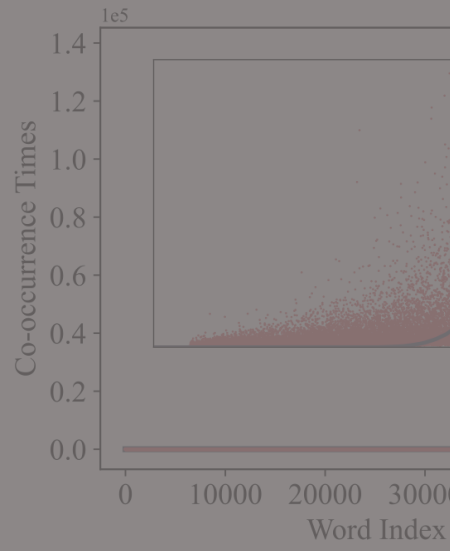


Key Observation: a small set of tokens are important along the generation

- accumulated attention scores of all the tokens follow a power-law distribution

Heavy-Hitters for Low Miss Rate

Challenge: how to evict to

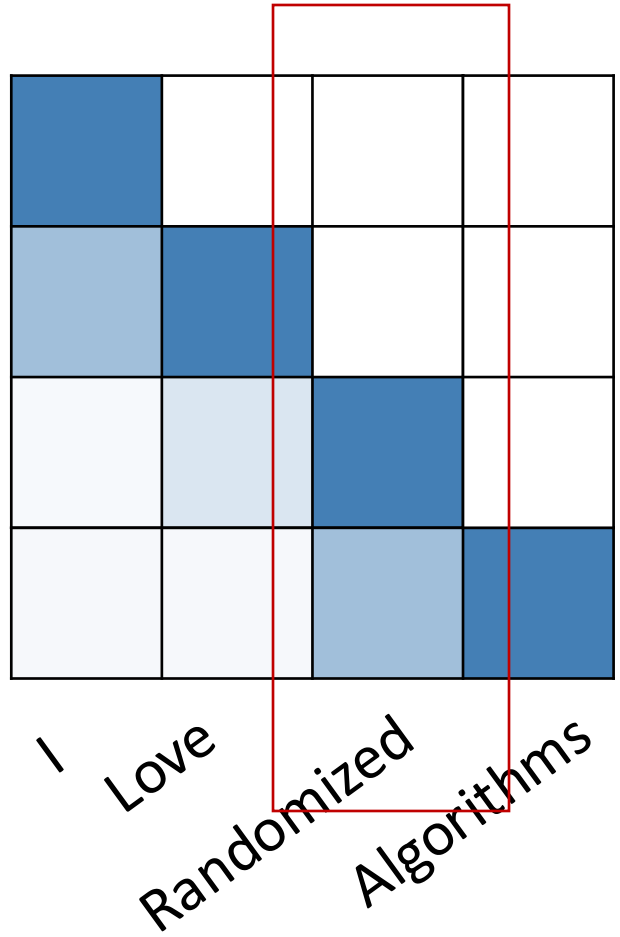


Key Observation: a small s

- accumulated attentic

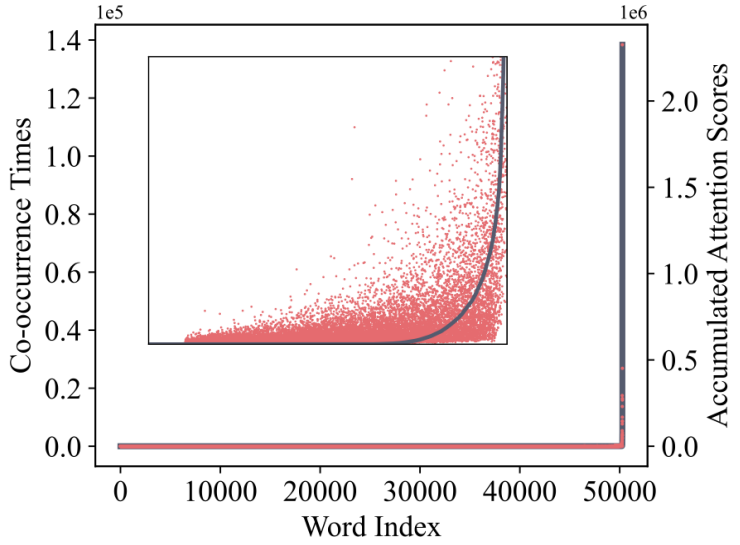
Q

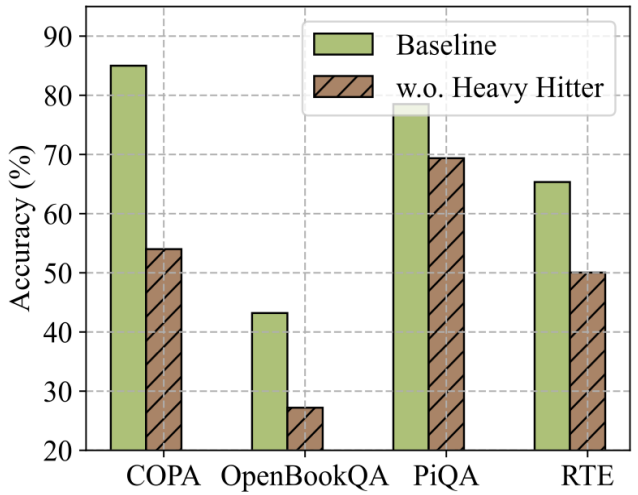
K cache



Heavy-Hitters for Low Miss Rate

Challenge: how to evict tokens? Once evicted, future tokens can no longer attend to it



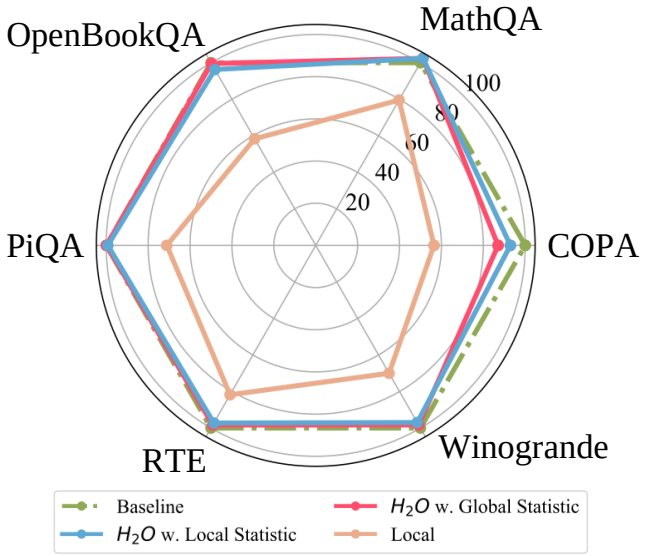


Key Observation: a small set of tokens are important along the generation

- accumulated attention scores of all the tokens follow a power-law distribution
- masking heavy-hitter tokens degrades model quality

Greedy Algorithm for Low-cost Policy

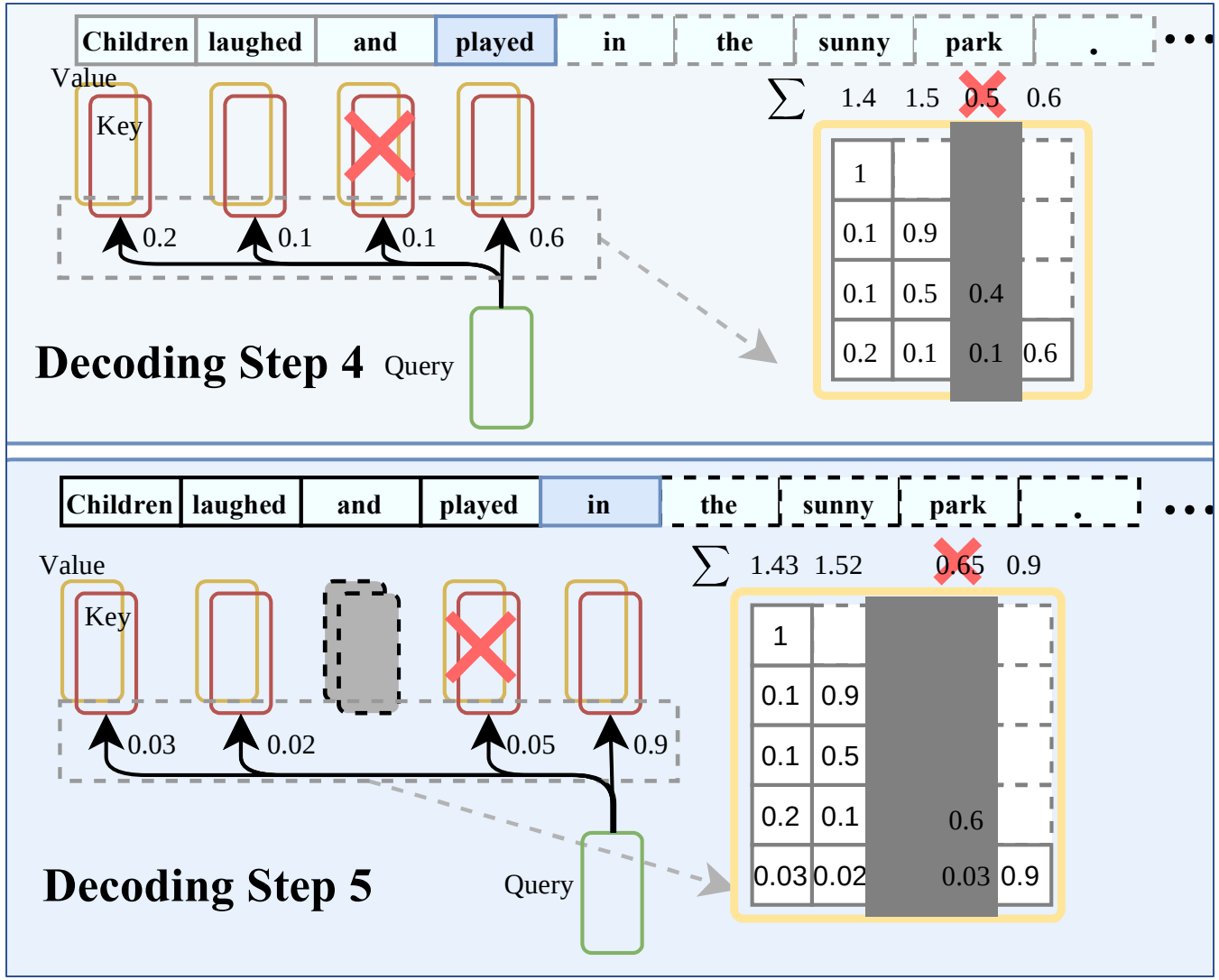
Challenge: how to deploy such algorithm without access to the full attention?



Idea: local greedy algorithm

- sum up the attention scores of the previous tokens every decoding step
- Add local / recent tokens

H_2O: Heavy Hitter Oracle



Model Input

In a small, bustling cafe nestled in the heart of a vibrant city, a serendipitous event unfolded, leaving a lasting impression on all who witnessed it. As the patrons sat sipping their coffees and engaging in animated conversations, a talented street musician entered the cafe, carrying a weathered guitar and radiating an aura of creativity.

LLaMA-7B Full Cache Output

He began to play, and the patrons were captivated. The musician's performance was so moving that the patrons began to applaud, and the musician was so moved that he began to cry. The patrons were so moved that they began to cry, and the musician was so

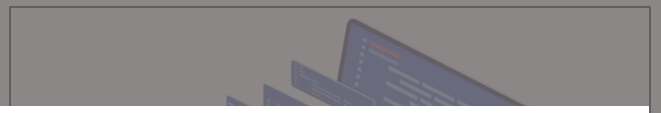
LLaMA-7B Local 20% Cache Output

He (((((((((((((, [)), 1999, 2000, 2001, 1, and, and, and, and, and, and, and, and, and, and, and, and,

LLaMA-7B H_2O 20% Cache Output

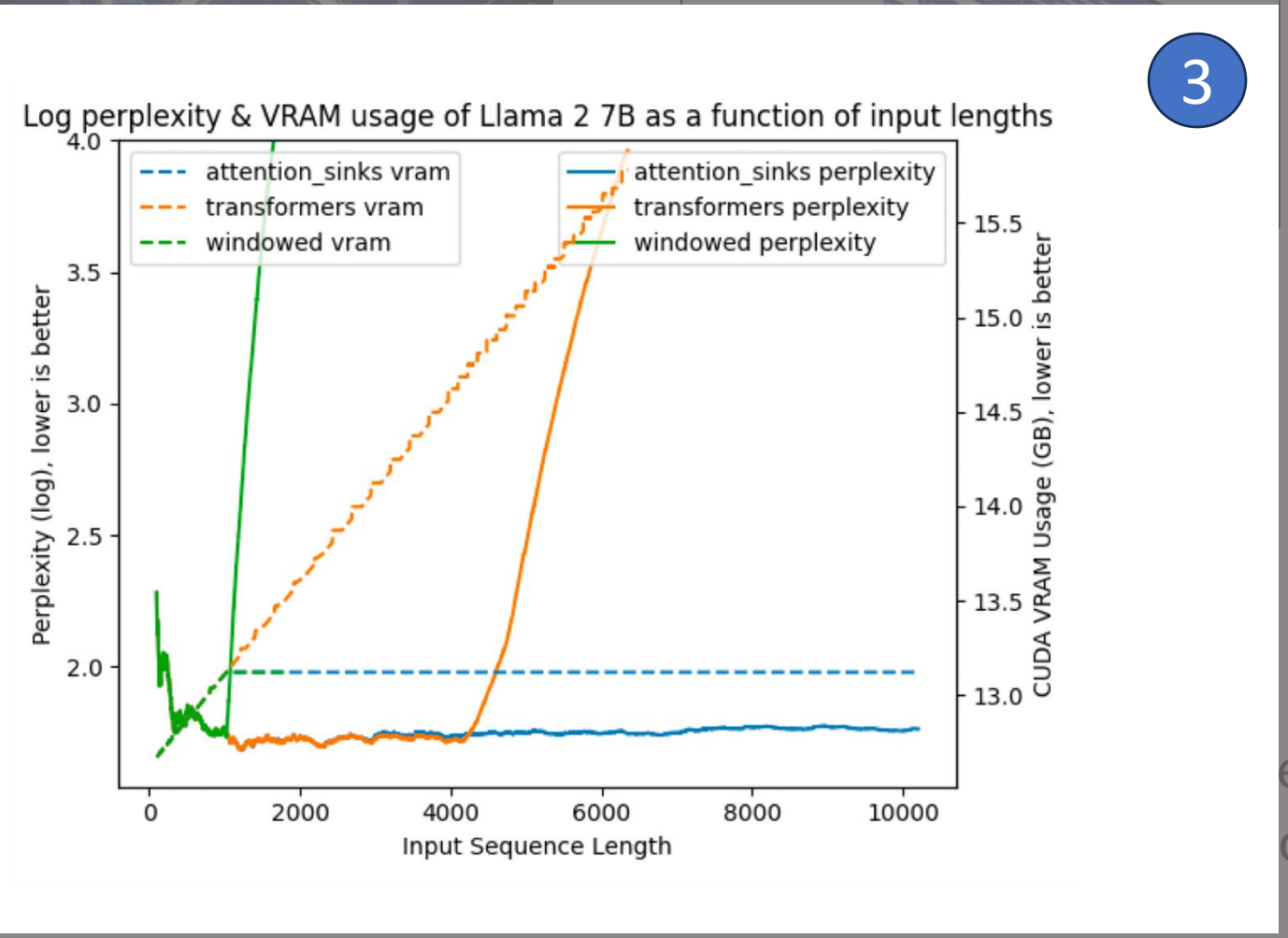
He began to play, and the room was filled with the sound of his music. The patrons of the cafe were enthralled by the music, and the atmosphere was electric. The cafe was packed with people, all of whom were enjoying the music. The musician was a young

LLMs are Powerful , but Very Expensive to Deploy



AI Agents

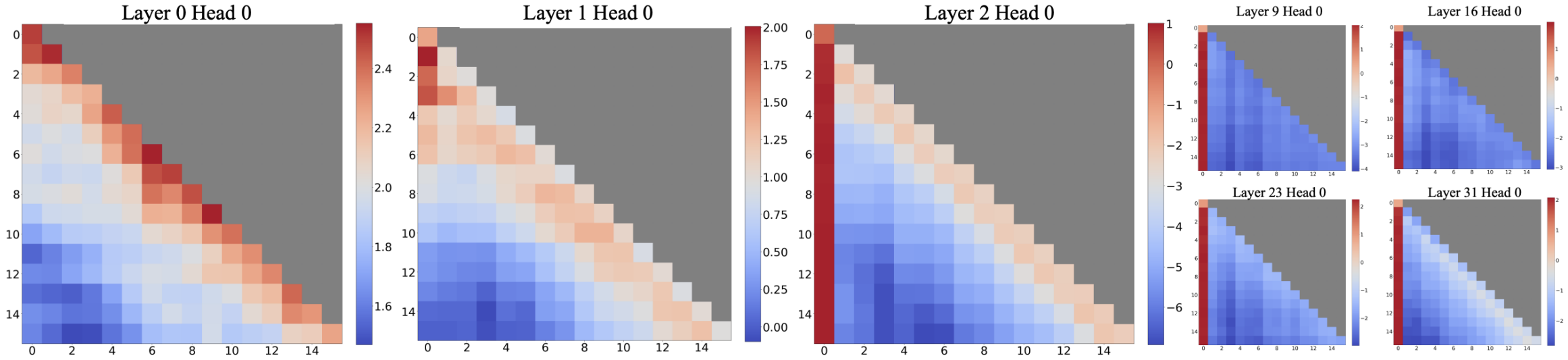
3



en + 64 batch size)

oad model, KV cache 100 times

Phenomenon: Attention Sink



Average attention logits in Llama-2-7B over 256 sentences

First few tokens!

- Observation: large attention scores are given to initial tokens, even if they're not semantically significant.
- Attention Sink: Tokens that disproportionately attract attention irrespective of their relevance.

Understanding Attention Sinks

- SoftMax operation's role in creating attention sinks — attention scores have to sum up to one for all contextual tokens. (*SoftMax-Off-by-One, Miller et al. 2023*)

$$\text{SoftMax}(x)_i = \frac{e^{x_i}}{e^{x_1} + \sum_{j=2}^{N} e^{x_j}}, \quad x_1 \gg x_j, j \in 2, \ldots, N$$

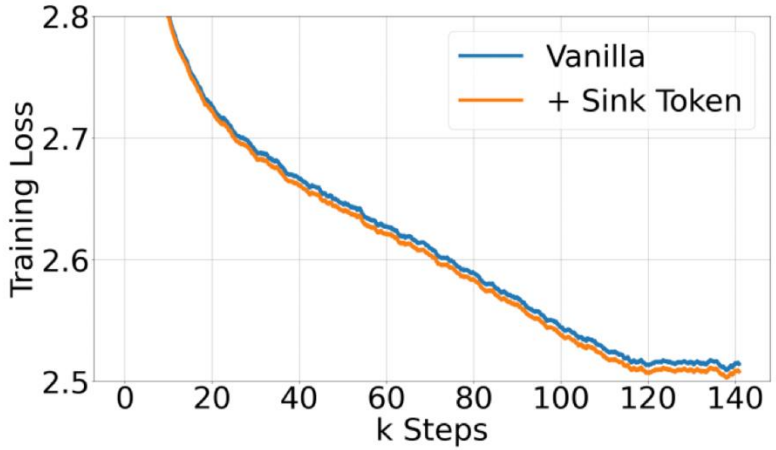
- Initial tokens' advantage in becoming sinks due to their visibility to subsequent tokens, rooted in autoregressive language modeling.

- The model learns a bias towards their absolute position rather than the semantics are crucial.

Llama-2-13B	PPL (↓)
0+1024 (window)	5158.07
4+1024	5.40
4”\n”+1020	5.6

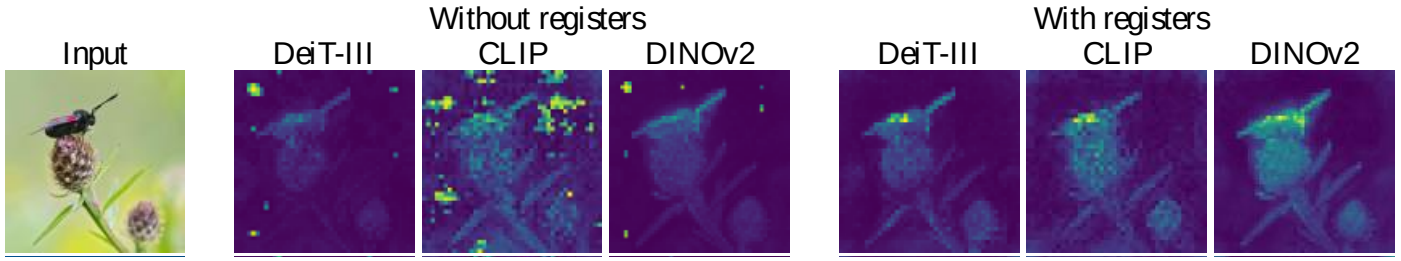
Understanding Attention Sinks

- Pre-train with a Dedicated Attention Sink Token

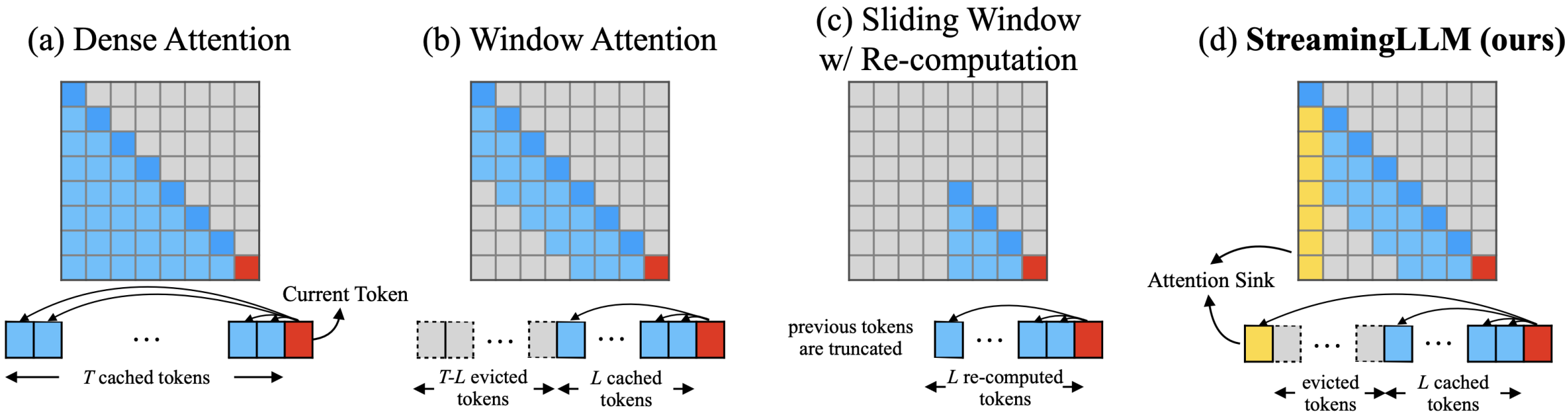


Cache Config	0+1024	1+1023	2+1022	4+1020
Vanilla	27.87	18.49	18.05	18.05
Zero Sink	29214	19.90	18.27	18.01
Learnable Sink	1235	**18.01**	18.01	18.02

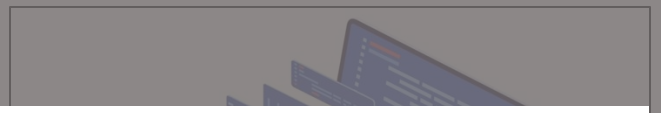
- Similar Phenomenon in *Darcet et al. Vision transformers need registers*



StreamingLLM

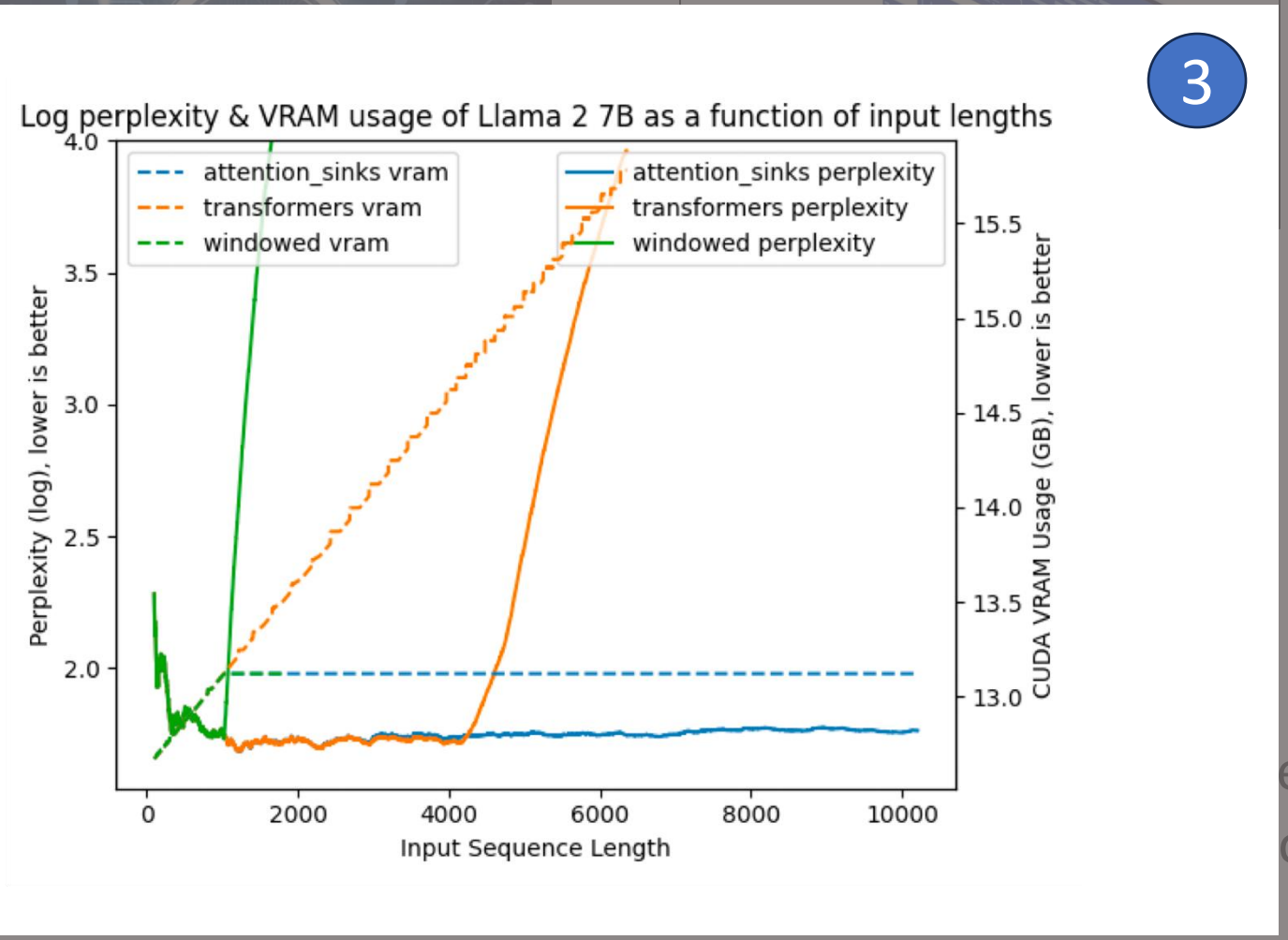


LLMs are Powerful , but Very Expensive to Deploy



AI Agents

3



en + 64 batch size)

oad model, KV cache 100 times

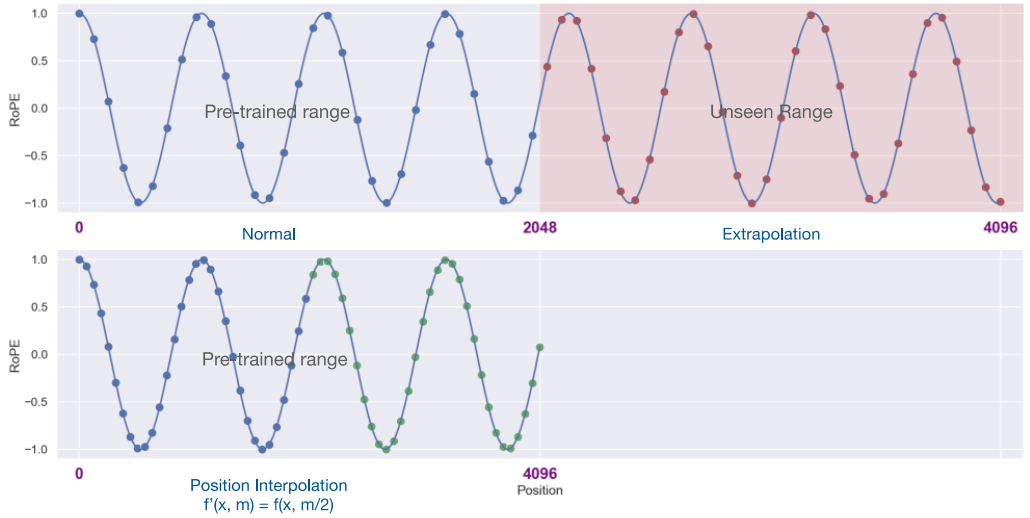
Position Interpolation

EXTENDING CONTEXT WINDOW OF LARGE LANGUAGE MODELS VIA POSITION INTERPOLATION

Shouyuan Chen **Sherman Wong** **Liangjian Chen** **Yuandong Tian**
Meta Platforms Inc.
{chenshouyuan,shermanwong,clj,yuandong}@meta.com

ABSTRACT

We present Position Interpolation (PI) that extends the context window sizes of RoPE-based (Su et al., 2021) pretrained LLMs such as LLaMA (Touvron et al., 2023) models to up to 32768 with minimal fine-tuning (within 1000 steps), while demonstrating strong empirical results on various tasks that require long context, including passkey retrieval, language modeling, and long document summarization from LLaMA 7B to 65B. Meanwhile, the extended model by Position Interpolation preserve quality relatively well on tasks within its original context window. To achieve this goal, Position Interpolation linearly down-scales the input position indices to match the original context window size, rather than extrapolating beyond the trained context length which may lead to catastrophically high attention scores that completely ruin the self-attention mechanism. Our theoretical study shows that the upper bound of interpolation is at least $\sim 600\times$ smaller than that of extrapolation, further demonstrating its stability. Models extended via Position Interpolation retain its original architecture and can reuse most pre-existing optimization and infrastructure.



Infinite Streaming Ability

Urgent need for LLMs in streaming applications such as multi-round dialogues, where long interactions are needed.

Key challenge:

- Pre-trained model (e.g., LLaMA) cannot go beyond its pre-trained context window

Train: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

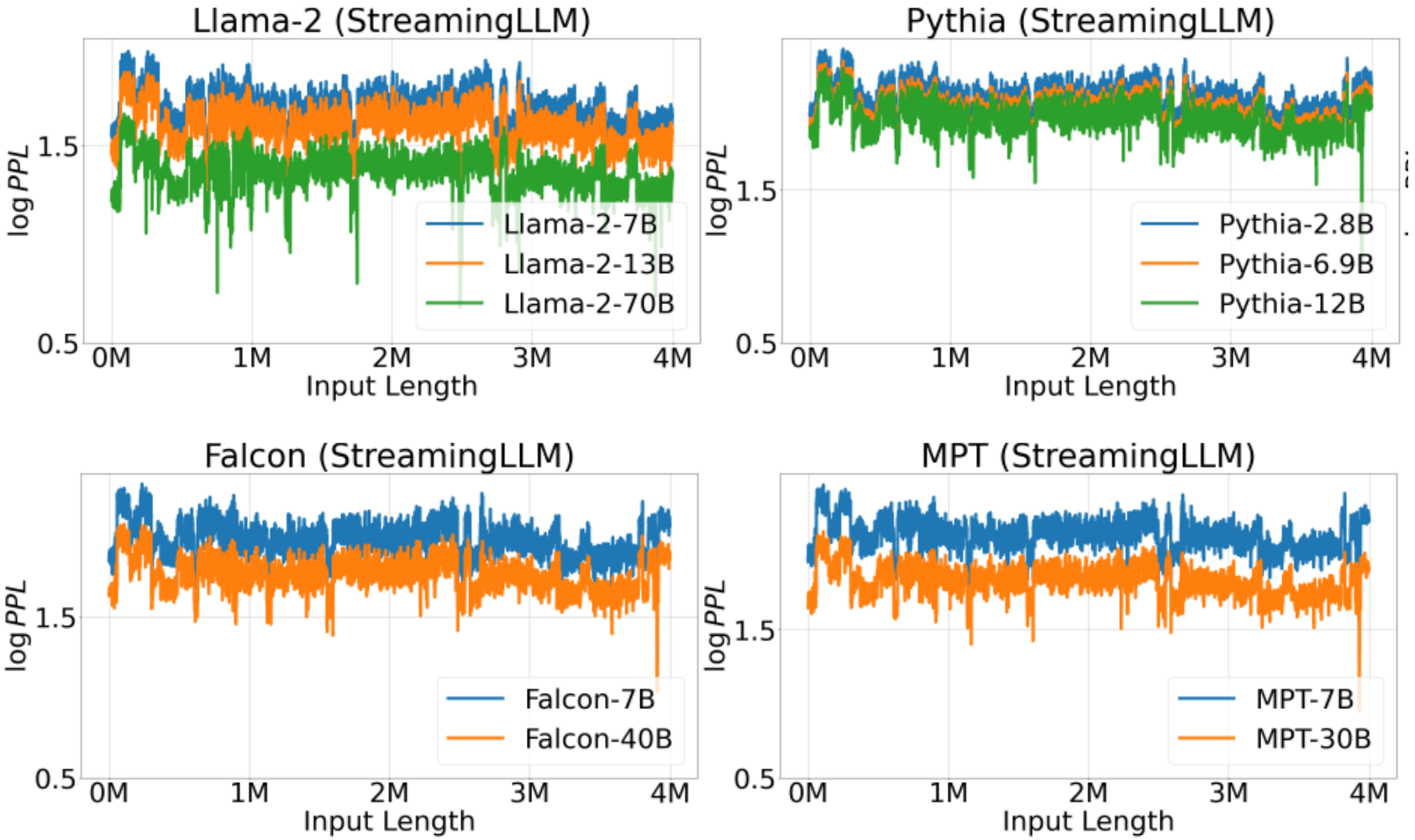
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | ? | ? |

Opportunity with StreamingLLM:

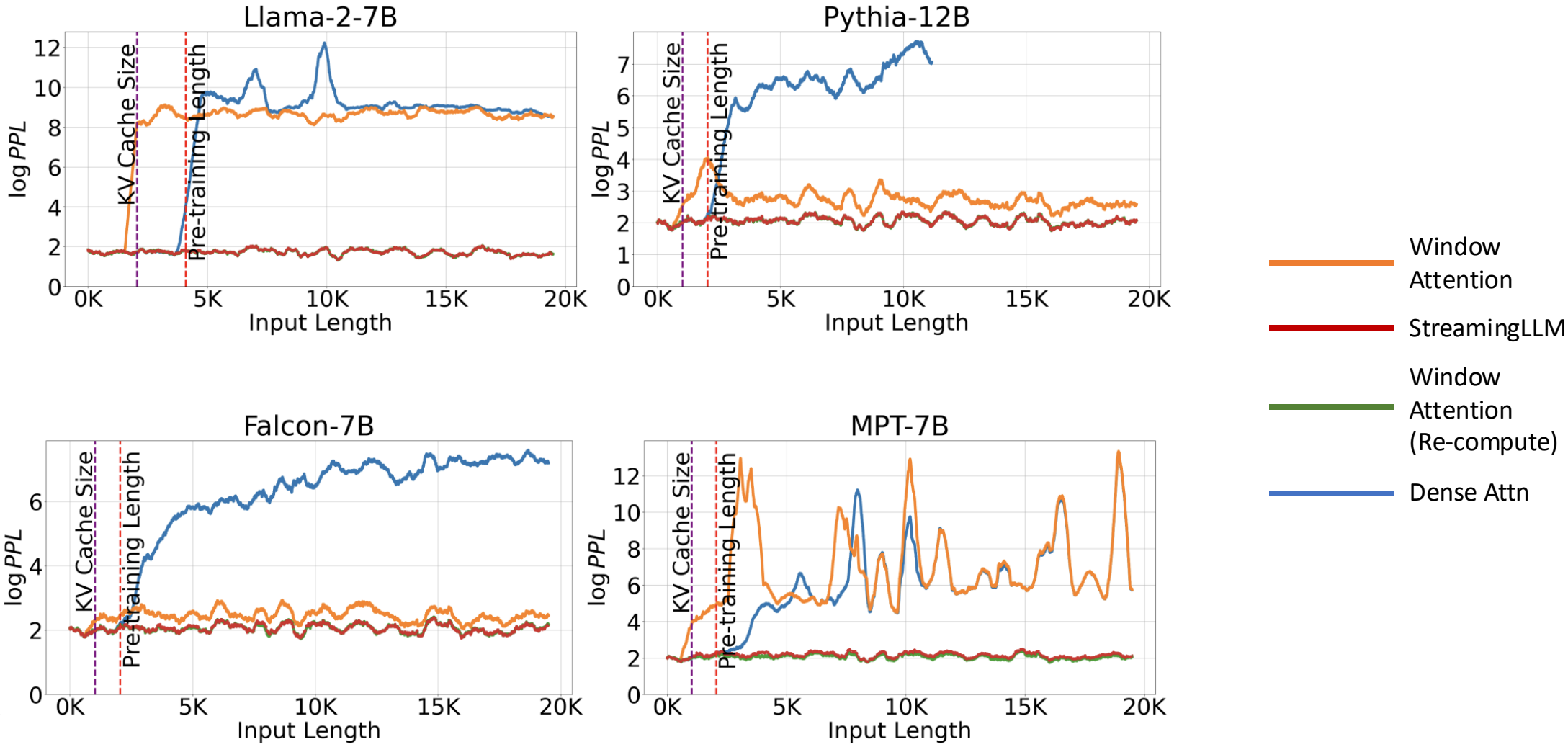
Train: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

| 1 | 2 | 3 | 4 | x | x | 5 | 6 | 7 | 8 |

Stably Model up to 4 Million Tokens



22X Faster than Sliding Window Recomputation



Infinite Streaming Ability

Urge LLMs in streaming applications such as multi-round dialogues, where long inter needed.

But StreamingLLM will forget the middle contents?

Key challenge:

- Pre-trained model (e.g., LLaMA) cannot go beyond its pre-trained context window

Train: 1 2 3 4 5 6 7 8 — 1 2 3 4 5 6 7 8 ? ?

Opportunity with StreamingLLM:

Train: 1 2 3 4 5 6 7 8 — 1 2 3 4 x x 5 6 7 8

The perplexity remains stable throughout up to 4 Million Tokens!

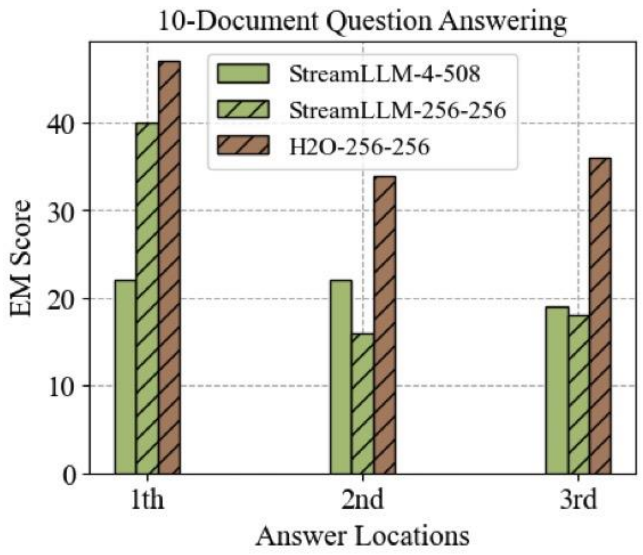
StreamingH2O: Infinite Streaming Ability

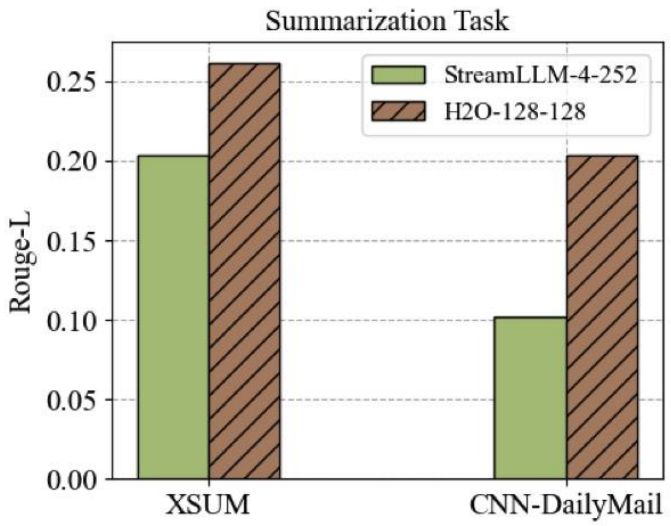
Similar position squeezing can be deployed on H2O

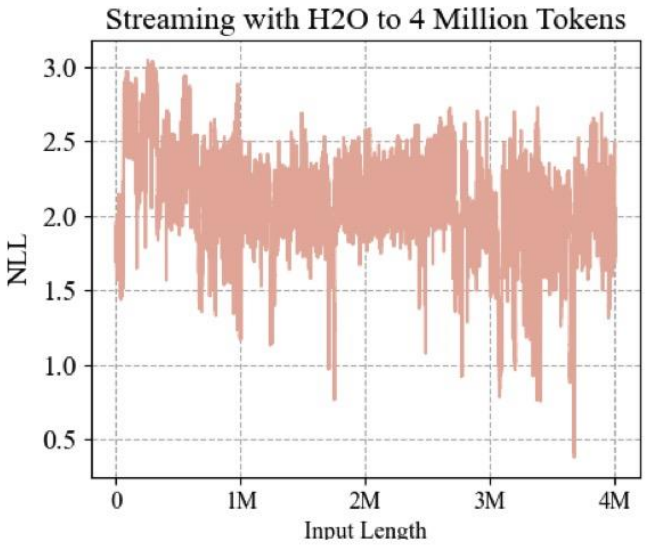
Train:

1	2	3	4	5	6	7	8

1	x	2	x	3	4	5	6	7	8







Serving on Commodity Hardware

FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU

Ying Sheng [1] Lianmin Zheng [2] Binhang Yuan [3] Zhuohan Li [2] Max Ryabinin [45] Daniel Y. Fu [1] Zhiqiang Xie [1]
Beidi Chen [67] Clark Barrett [1] Joseph E. Gonzalez [2] Percy Liang [1] Christopher Ré [1] Ion Stoica [2] Ce Zhang [3]

Abstract

The high computational and memory requirements of large language model (LLM) inference make it feasible only with multiple high-end accelerators. Motivated by the emerging demand for latency-insensitive tasks with batched processing, this paper initiates the study of high-throughput LLM inference using limited resources, such as a single commodity GPU. We present FlexGen, a high-throughput generation engine for running LLMs with limited GPU memory. FlexGen can be flexibly configured under various hardware resource constraints by aggregating memory and computation from the GPU, CPU, and disk. By solving a linear programming problem, it searches

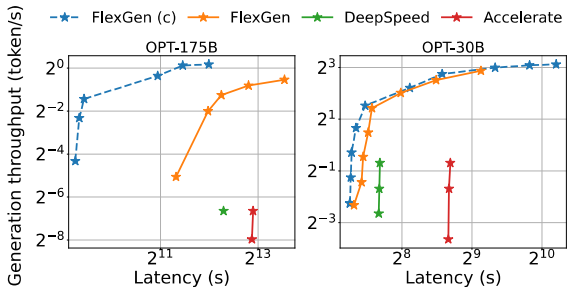
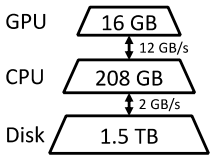
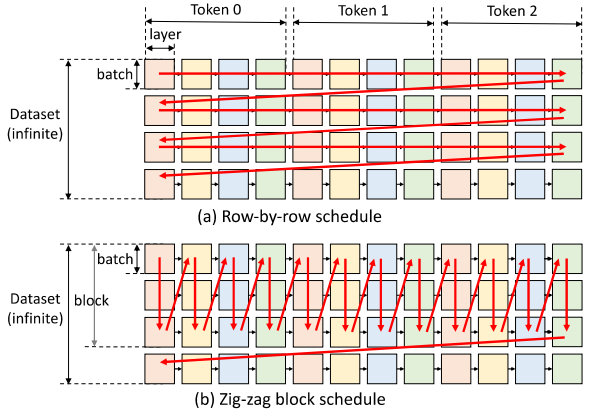


Figure 1. The total latency for a block and throughput trade-offs of three offloading-based systems for OPT-175B (left) and OPT-30B (right) on a single NVIDIA T4 (16 GB) GPU with 208 GB CPU DRAM and 1.5TB SSD. FlexGen achieves a new Pareto-optimal frontier with 100× higher maximum throughput for OPT-175B. Other systems cannot further increase throughput due to out-of-memory issues. "(c)" denotes compression.

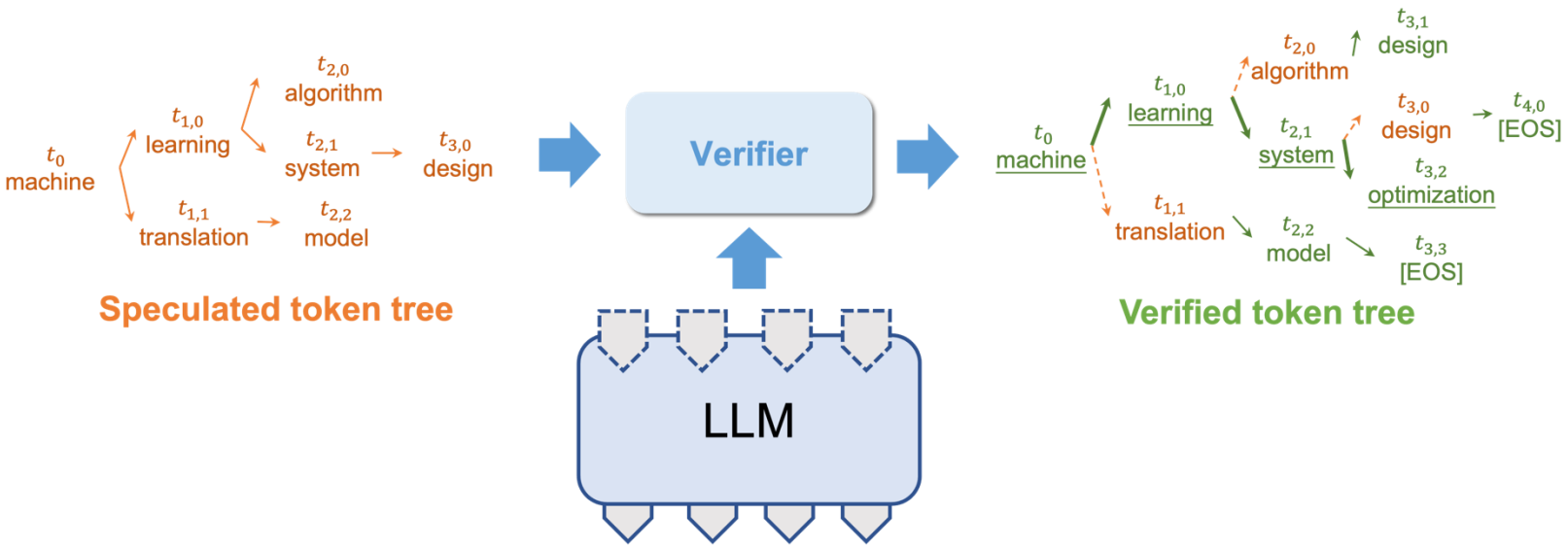




Serving in Latency Constraint Setting

The idea of speculative decoding to accelerate LLM inference while preserving the LLM's output distribution has been widely studied!

- (*Chen et al 2023, Leviathan et al 2023, SpecInfer, SpecTr, …*)



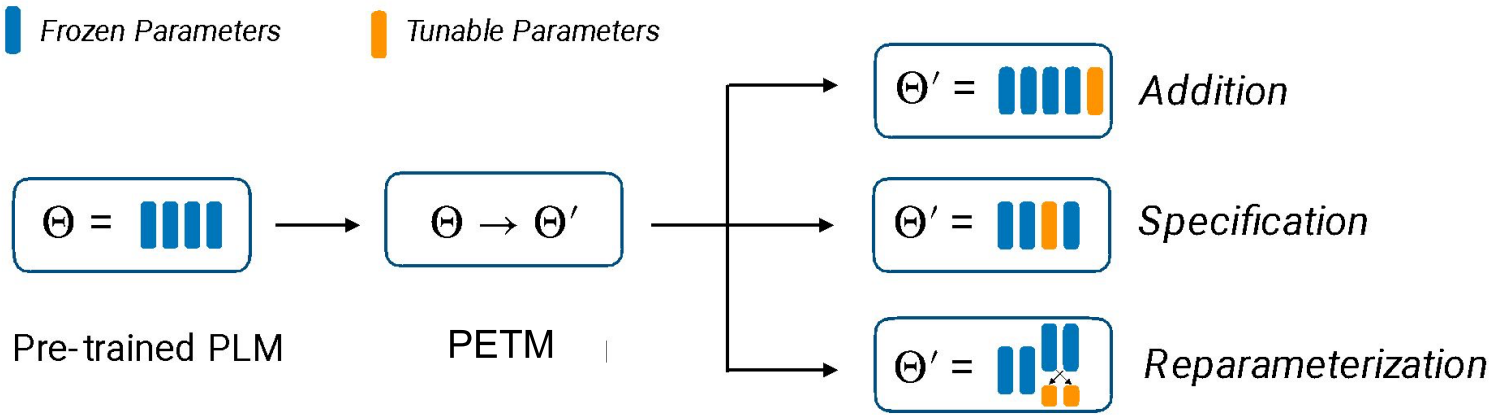
*SpecInfer

Why parameter-efficient tuning?

- Prompt engineering
  - Doesn’t always work
  - Tedious to find a good prompt
- Finetune the full LM
  - Expensive
  - Overfitting / catastrophic forgetting on small datasets
  - Need to store one full set of model weights per task
- PEFT
  - Rather than finetuning the entire model, we finetune only small amounts of weights.

PEFT

- Addition: What if we introduce additional trainable parameters to the neural network and just train those?
- Specification: What if we pick a small subset of the parameters of the neural network and just tune those?
- Reparameterization: What if we re-parameterize the model into something that is more efficient to train?



Prompt Tuning

The Power of Scale for Parameter-Efficient Prompt Tuning

Brian Lester* Rami Al-Rfou Noah Constant
Google Research
{brianlester,rmyeid,nconstant}@google.com

Abstract

In this work, we explore "prompt tuning," a simple yet effective mechanism for learning "soft prompts" to condition frozen language models to perform specific downstream tasks. Unlike the discrete text prompts used by GPT-3, soft prompts are learned through backpropagation and can be tuned to incorporate signals from any number of labeled examples. Our end-to-end learned approach outperforms GPT-3's few-shot learning by a large margin. More remarkably, through ablations on model size using T5, we show that prompt tuning becomes more competitive with scale: as models exceed billions of parameters, our method "closes the gap" and matches the strong performance of model tuning (where all model weights are tuned). This finding is especially relevant because large models are costly to share and serve and the ability to reuse one frozen model for multiple downstream tasks can ease this burden. Our method can be seen

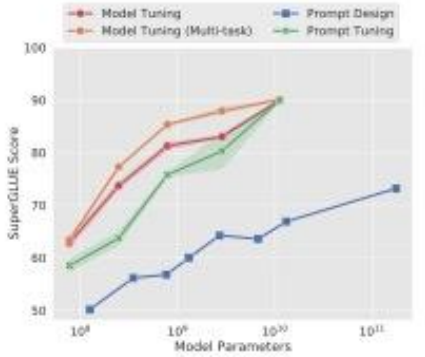
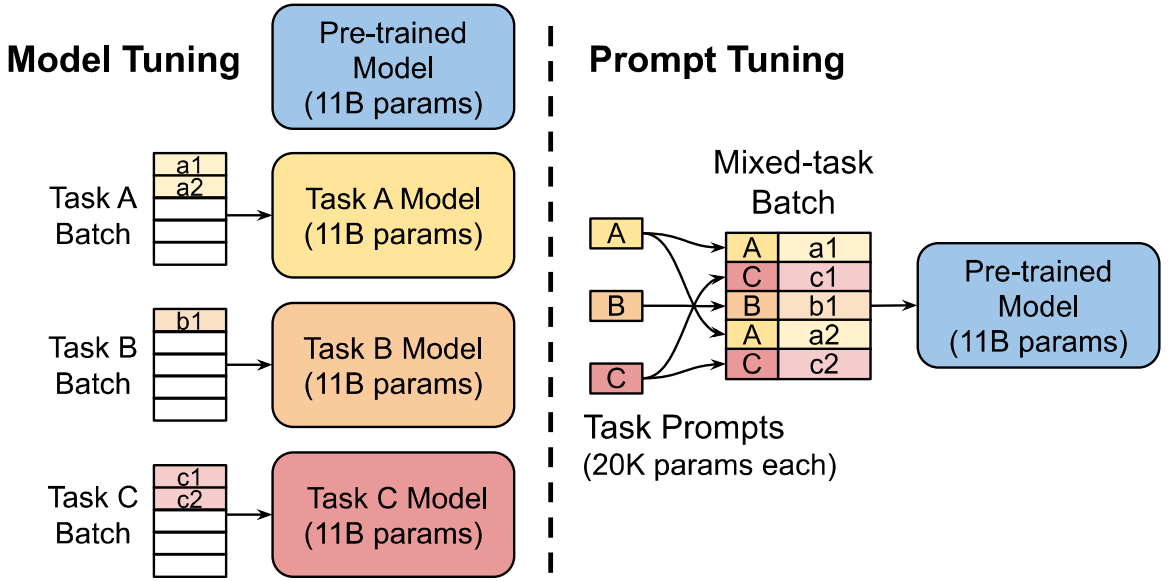


Figure 1: Standard **model tuning** of T5 achieves strong performance, but requires storing separate copies of the model for each end task. Our **prompt tuning** of T5 matches the quality of model tuning as size increases, while enabling the reuse of a single frozen model for all tasks. Our approach significantly outperforms few-shot **prompt design** using GPT-3. We show mean and standard deviation across 3 runs for tuning methods.



Prompt Tuning

The Power of Scale for Parameter-Efficient Prompt Tuning

Brian Lester* Rami Al-Rfou Noah Constant
Google Research
{brianlester,rmyeid,nconstant}@google.com

Abstract

In this work, we explore "prompt tuning," a simple yet effective mechanism for learning "soft prompts" to condition frozen language models to perform specific downstream tasks. Unlike the discrete text prompts used by GPT-3, soft prompts are learned through backpropagation and can be tuned to incorporate signals from any number of labeled examples. Our end-to-end learned approach outperforms GPT-3's few-shot learning by a large margin. More remarkably, through ablations on model size using T5, we show that prompt tuning becomes more competitive with scale: as models exceed billions of parameters, our method "closes the gap" and matches the strong performance of model tuning (where all model weights are tuned). This finding is especially relevant because large models are costly to share and serve and the ability to reuse one frozen model for multiple downstream tasks can ease this burden. Our method can be seen

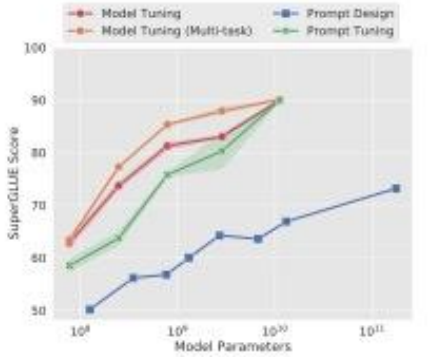


Figure 1: Standard **model tuning** of T5 achieves strong performance, but requires storing separate copies of the model for each end task. Our **prompt tuning** of T5 matches the quality of model tuning as size increases, while enabling the reuse of a single frozen model for all tasks. Our approach significantly outperforms few-shot **prompt design** using GPT-3. We show mean and standard deviation across 3 runs for tuning methods.

- Finetune T5 to act a bit more like a traditional language model.
  - This only needs to be done once, and empirically makes prompt tuning working better.
  - This is probably because the span-corruption objective T5 was originally trained with isn't amenable to prompting.
- Freeze the weights of T5. Set the first k input embeddings to be learnable.
  - k is a hyperparameter up to the choice of the implementer.
- Initialize the k learnable embeddings. Some options include:
  - Random initialization
  - Initialize to values drawn from the vocabulary embedding matrix.
- Train on your task specific data

Prefix Tuning

Prefix-Tuning: Optimizing Continuous Prompts for Generation

Xiang Lisa Li
Stanford University
xlisali@stanford.edu

Percy Liang
Stanford University
pliang@cs.stanford.edu

Abstract

Fine-tuning is the de facto way of leveraging large pretrained language models for downstream tasks. However, fine-tuning modifies all the language model parameters and therefore necessitates storing a full copy for each task. In this paper, we propose prefix-tuning, a lightweight alternative to fine-tuning for natural language generation tasks, which keeps language model parameters frozen and instead optimizes a sequence of *continuous task-specific* vectors, which we call the *prefix*. Prefix-tuning draws inspiration from prompting for language models, allowing subsequent tokens to attend to this prefix as if it were "virtual tokens". We apply prefix-tuning to GPT-2 for table-to-text generation and to BART for summarization. We show that by modifying only 0.1% of the parameters, prefix-tuning obtains comparable performance in the full data setting, outperforms fine-tuning in low-data settings, and extrapolates better to examples with topics that

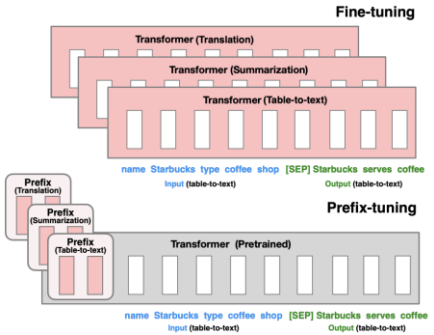
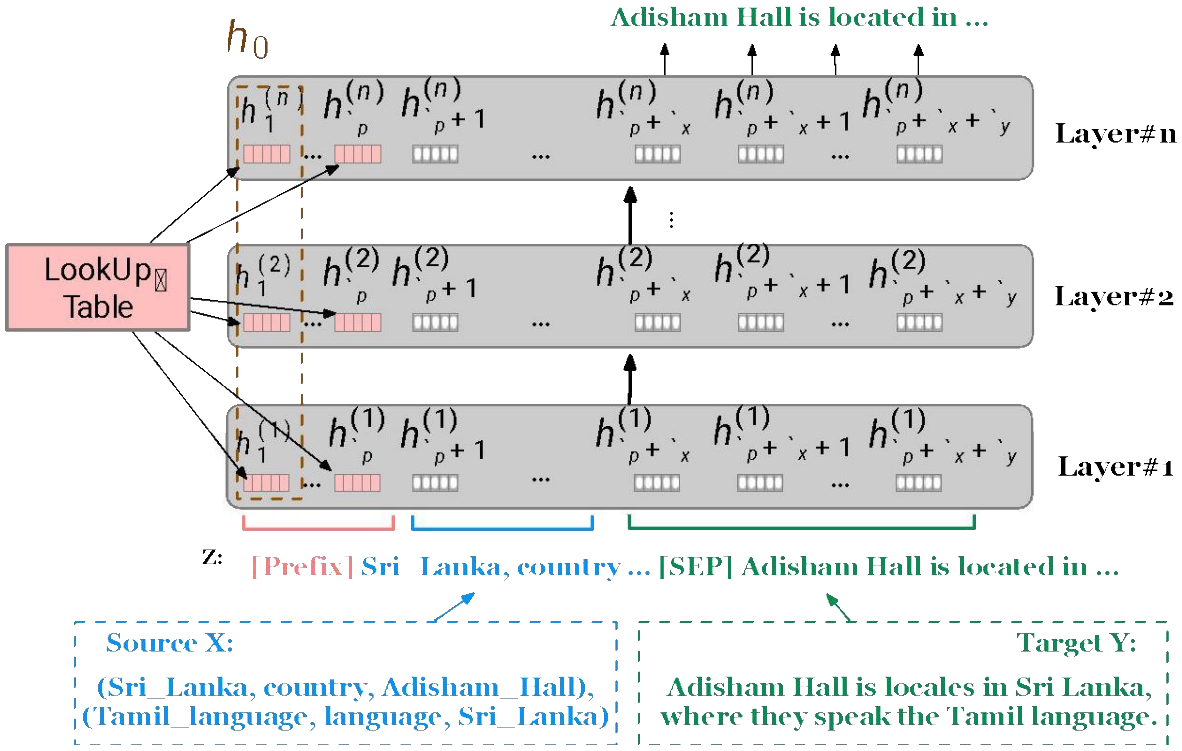


Figure 1: Fine-tuning (top) updates all LM parameters (the red Transformer box) and requires storing a full model copy for each task. We propose prefix-tuning (bottom), which freezes the LM parameters and only optimizes the prefix (the red prefix blocks). Consequently, we only need to store the prefix for each task, making prefix-tuning modular and space-efficient. Note that each vertical block denote transformer activations at one time step.



Advantages / Disadvantages

- The learned embeddings tend to be relatively small, just a few megabytes or less.
  - It is cheap to keep around one set of embeddings per task.
- The pre-trained LLM can be loaded into memory (such as on a server), and at inference time, the appropriate task-specific embeddings can be passed in.
  - Example use case: User customization

- In practice, these methods tend to converge significantly slower than full parameter fine-tuning.
- Unclear what the best prefix length is for any particular task.
  - Every sequence position you “spend” on the prefix is one less you have for your actual task.
- Learned embeddings are not very interpretable.

Adapters

Parameter-Efficient Transfer Learning for NLP

Neil Houlsby [1] Andrei Giurgiu [1 *] Stanisław Jastrzębski [2 *] Bruna Morrone [1] Quentin de Laroussilhe [1]
Andrea Gesmundo [1] Mona Attariyan [1] Sylvain Gelly [1]

Abstract

Fine-tuning large pre-trained models is an effective transfer mechanism in NLP. However, in the presence of many downstream tasks, fine-tuning is parameter inefficient: an entire new model is required for every task. As an alternative, we propose transfer with adapter modules. Adapter modules yield a compact and extensible model; they add only a few trainable parameters per task, and new tasks can be added without revisiting previous ones. The parameters of the original network remain fixed, yielding a high degree of parameter sharing. To demonstrate adapter's effectiveness, we transfer the recently proposed BERT Transformer model to 26 diverse text classification tasks, including the GLUE benchmark. Adapters attain near state-of-the-art performance, whilst adding only a few parameters per task. On GLUE, we attain within 0.4% of the performance of full fine-tuning, adding only 3.6% parameters per task. By contrast, fine-tuning trains 100% of the parameters per task.[1]

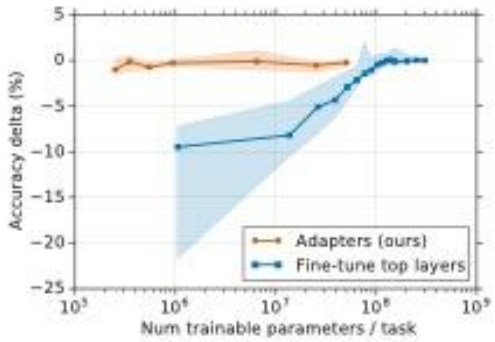
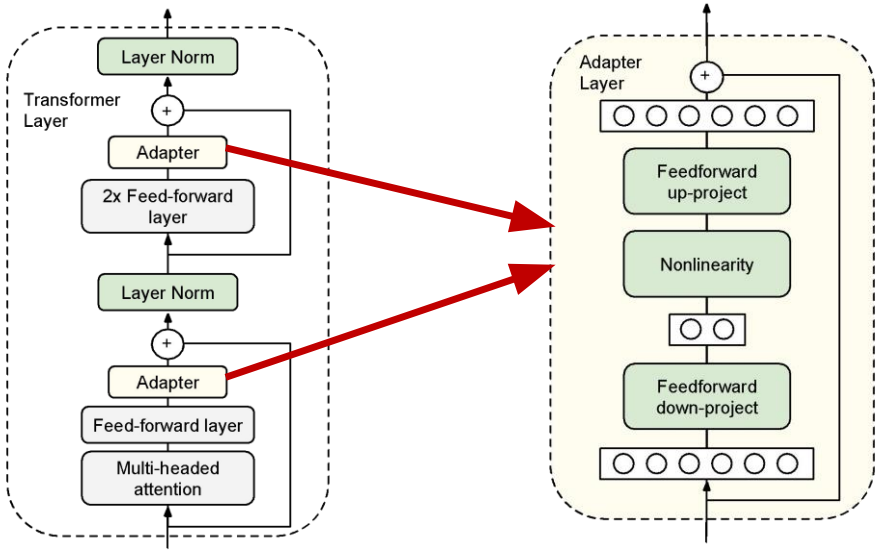


Figure 1. Trade-off between accuracy and number of trained task-specific parameters, for adapter tuning and fine-tuning. The y-axis is normalized by the performance of full fine-tuning, details in Section 3. The curves show the 20th, 50th, and 80th performance percentiles across nine tasks from the GLUE benchmark. Adapter-based tuning attains a similar performance to full fine-tuning with two orders of magnitude fewer trained parameters.



Compacters

COMPACTER: Efficient Low-Rank Hypercomplex Adapter Layers

Rabeeh Karimi Mahabadi
EPFL University, Idiap Research Institute
rabeeh.karimi@idiap.ch

James Henderson
Idiap Research Institute
james.henderson@idiap.ch

Sebastian Ruder
DeepMind
ruder@google.com

Abstract

Adapting large-scale pretrained language models to downstream tasks via fine-tuning is the standard method for achieving state-of-the-art performance on NLP benchmarks. However, fine-tuning all weights of models with millions or billions of parameters is sample-inefficient, unstable in low-resource settings, and wasteful as it requires storing a separate copy of the model for each task. Recent work has developed *parameter-efficient* fine-tuning methods, but these approaches either still require a relatively large number of parameters or underperform standard fine-tuning. In this work, we propose COMPACTER, a method for fine-tuning large-scale language models with a better trade-off between task performance and the number of trainable parameters than prior work. COMPACTER accomplishes this by building on top of ideas from adapters, low-rank optimization, and parameterized hypercomplex multiplication layers.

Specifically, COMPACTER inserts task-specific weight matrices into a pretrained model's weights, which are computed efficiently as a sum of Kronecker products between shared "slow" weights and "fast" rank-one matrices defined per COMPACTER layer. By only training 0.047% of a pretrained model's parameters, COMPACTER performs on par with standard fine-tuning on GLUE and outperforms standard fine-tuning on SuperGLUE and low-resource settings. Our code is publicly available at https://github.com/rabeehk/compacter

- **Compacters** are an extension of adapters which aim to make the technique even more efficient.
- Adapters are standard fully connected layers.
  - Linear project to lower dimension followed by nonlinearity followed by projection back up to original dimension.
  - $y = \boldsymbol{W_2}\ \mathrm{GELU}(\boldsymbol{W_1 x} + \boldsymbol{b_1}) + \boldsymbol{b_2}$
- The compacter replaces the fully connected layer with a. parameterized hypercomplex multiplication layer.
  - Each $\boldsymbol{W}$ is learned as a sum of n Kronecker products
  - n is a user-specified hyperparameter.
- Compacters reduce the number of parameters in the adapter layer to $1/n$ without harming the performance.

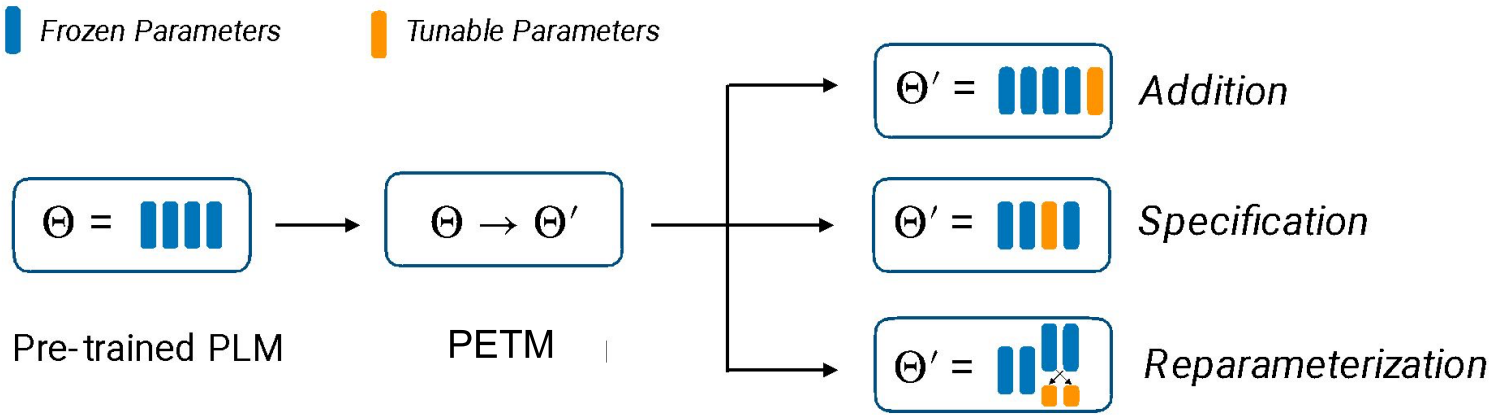
Advantages / Disadvantages

- Have been shown to be quite effective in multi-task settings.
  - There are methods for training task-specific adapters and then combining the to leverage the cross-task knowledge• Tend to be faster to tune than full model finetuning.
- Possibly more robust to adversarial perturbations of the tuning data than full model finetuning.

- Adding in new layers means making inference slower.
- It also makes the model bigger (possibly harder to fit on available GPUs).
- Adapter layers need to be processed sequentially at inference time, which can break model parallelism.

PEFT

- Addition: What if we introduce additional trainable parameters to the neural network and just train those?
- **Specification: What if we pick a small subset of the parameters of the neural network and just tune those?**
- Reparameterization: What if we re-parameterize the model into something that is more efficient to train?



Layer Freezing

BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models

Elad Ben-Zaken[1] **Shauli Ravfogel**[1,2] **Yoav Goldberg**[1,2]
[1]Computer Science Department, Bar Ilan University
[2]Allen Institute for Artificial Intelligence
{benzakenelad, shauli.ravfogel, yoav.goldberg}@gmail.com

Abstract

We introduce BitFit, a sparse-finetuning method where only the bias-terms of the model (or a subset of them) are being modified. We show that with small-to-medium training data, applying BitFit on pre-trained BERT models is competitive with (and sometimes better than) fine-tuning the entire model. For larger data, the method is competitive with other sparse fine-tuning methods. Besides their practical utility, these findings are relevant for the question of understanding the commonly-used process of finetuning: they support the hypothesis that finetuning is mainly about exposing knowledge induced by language-modeling training, rather than learning new task-specific linguistic knowledge.

3. The changed parameters are both isolated and localized across the entire parameter space.
4. For small to medium training data, changing only these parameters reaches the same task accuracy as full fine-tuning, and sometimes even improves results.

Specifically, we show that freezing most of the network and **fine-tuning only the bias-terms** is surprisingly effective. Moreover, if we allow the tasks to suffer a small degradation in performance, we can fine-tune only two bias components (the "query" and "middle-of-MLP" bias terms), amounting to half of the bias parameters in the model, and only 0.04% of all model parameters.

This result has a large practical utility in de-

- Only tune the bias terms and final classification layer (if doing classification)
- Recall the equations for multi-head attention

$$\mathbf{Q}^{m,\ell}(\mathbf{x}) = \mathbf{W}_q^{m,\ell}\mathbf{x} + \mathbf{b}_q^{m,\ell}$$
$$\mathbf{K}^{m,\ell}(\mathbf{x}) = \mathbf{W}_k^{m,\ell}\mathbf{x} + \mathbf{b}_k^{m,\ell}$$
$$\mathbf{V}^{m,\ell}(\mathbf{x}) = \mathbf{W}_v^{m,\ell}\mathbf{x} + \mathbf{b}_v^{m,\ell}$$

  - ℓ is the layer index
  - m is the attention head index
  - Only the bias terms (shown in red) are updated.

DiffPruning

Parameter-Efficient Transfer Learning with Diff Pruning

Demi Guo
Harvard University
dguo@college.harvard.edu

Alexander M. Rush
Cornell University
arush@cornell.edu

Yoon Kim
MIT CSAIL
MIT-IBM Watson AI
yoonkim@mit.edu

Abstract

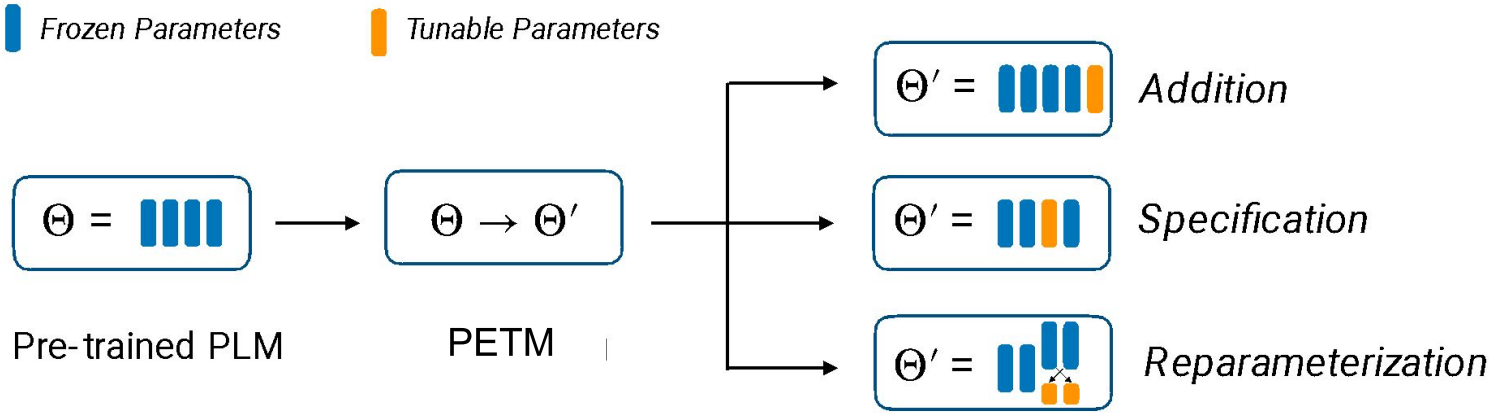
The large size of pretrained networks makes them difficult to deploy for multiple tasks in storage-constrained settings. *Diff pruning* enables parameter-efficient transfer learning that scales well with new tasks. The approach learns a task-specific "diff" vector that extends the original pretrained parameters. This diff vector is adaptively pruned during training with a differentiable approximation to the L_0-norm penalty to encourage sparsity. As the number of tasks increases, diff pruning remains parameter-efficient, as it requires storing only a small diff vector for each task. Since it does not require access to all tasks during training, it is attractive in on-device deployment settings where tasks arrive in stream or even from different providers. Diff pruning can match the performance of finetuned baselines on the GLUE benchmark while only modifying 0.5% of the pretrained model's pa

A popular approach to parameter-efficiency is to learn smaller compressed models for each task (Gordon et al., 2020; Sajjad et al., 2020; Zhao et al., 2020; Sanh et al., 2020). Such approaches face a steep sparsity/performance tradeoff and keep a substantial amount of nonzero parameters per task (e.g. 10%-30%). Multi-task learning and feature-based transfer allow for more parameter-efficient transfer learning per task (Liu et al., 2019b; Clark et al., 2019; Stickland & Murray, 2019; Reimers & Gurevych, 2019). These methods train a small number of additional parameters (e.g. a linear layer) on top of a shared model. However, multi-task learning generally requires access to all tasks during training to prevent catastrophic forgetting (French, 1999), while feature-based transfer learning (e.g. based on task-agnostic sentence representations) is typically outperformed by finetuning (Howard & Ruder, 2018).

- In prior methods we discussed, the choice of what parameters to freeze and what parameters to tune was done manually.
- Why not learn this instead?
- Main idea:
  - For each parameter, finetune a learnable "delta" which gets added to the original parameter value.
  - Use an L_0-norm penalty to encourage sparsity in the deltas.

PEFT

- Addition: What if we introduce additional trainable parameters to the neural network and just train those?
- Specification: What if we pick a small subset of the parameters of the neural network and just tune those?
- **Reparameterization: What if we re-parameterize the model into something that is more efficient to train?**



Intuition

- Finetuning has a low **intrinsic dimension**, that is, the minimum number of parameters needed to be modified to reach satisfactory performance is not very large.
- This means, we can reparameterize a subset of the original model parameters with low-dimensional proxy parameters, and just optimize the proxy.

- An objective function's intrinsic dimension measures the minimum number of parameters needed to reach a satisfactory solution to the objective.
- Can also be thought of as the lowest dimensional subspace in which one can optimize the original objective function to within a certain level of approximation error.

LoRA

LoRA: Low-Rank Adaptation of Large Language Models

Edward Hu* Yelong Shen* Phillip Wallis Zeyuan Allen-Zhu
Yuanzhi Li Shean Wang Lu Wang Weizhu Chen
Microsoft Corporation
{edwardhu, yeshe, phwallis, zeyuana,
yuanzhil, swang, luw, wzchen}@microsoft.com
yuanzhil@andrew.cmu.edu
(Version 2)

Abstract

An important paradigm of natural language processing consists of large-scale pre-training on general domain data and adaptation to particular tasks or domains. As we pre-train larger models, full fine-tuning, which retrains all model parameters, becomes less feasible. Using GPT-3 175B as an example – deploying independent instances of fine-tuned models, each with 175B parameters, is prohibitively expensive. We propose Low-Rank Adaptation, or LoRA, which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks. Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times. LoRA performs on-par or better than fine-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having fewer trainable parameters, a higher training throughput, and, unlike adapters, *no additional inference latency*. We also provide an empirical investigation into rank-deficiency in language model adaptation, which sheds light on the efficacy of LoRA. We release a package that facilitates the integration of LoRA with PyTorch models and provide our implementations and model checkpoints for RoBERTa, DeBERTa, and GPT-2 at https://github.com/microsoft/LoRA.

- Intuition: It's not just the model weights that are low rank, *updates* to the model weights are also low-rank.
- LoRA freezes the pretrained model weights and injects trainable rank decomposition matrices into each layer.
- Like DiffPruning, we are learning a delta to apply to each weight. In the case of LoRA, this delta has been re-paramaterized to be lower dimension than the original model parameters.
- In practice, LoRA only adapts the attention weights and keeps the rest of the Transformer as-is.

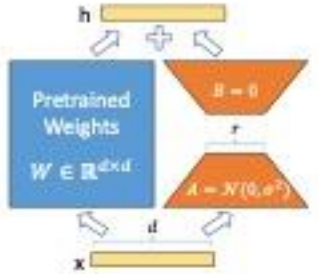


Figure 1: Our reparametrization. We only train A and B.

QLoRA

QLoRA: Efficient Finetuning of Quantized LLMs

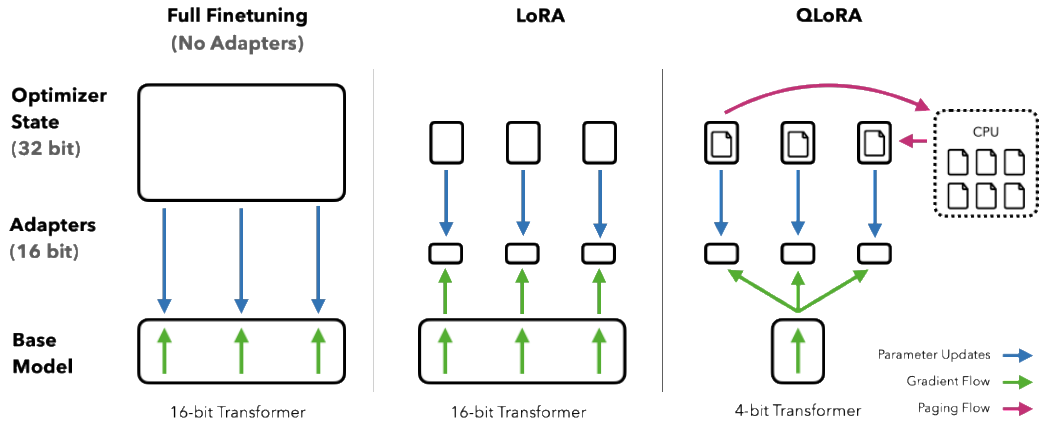
Tim Dettmers* **Artidoro Pagnoni*** **Ari Holtzman**

Luke Zettlemoyer

University of Washington
{dettmers,artidoro,ahai,lsz}@cs.washington.edu

Abstract

We present QLoRA, an efficient finetuning approach that reduces memory usage enough to finetune a 65B parameter model on a single 48GB GPU while preserving full 16-bit finetuning task performance. QLoRA backpropagates gradients through a frozen, 4-bit quantized pretrained language model into Low Rank Adapters (LoRA). Our best model family, which we name **Guanaco**, outperforms all previous openly released models on the Vicuna benchmark, reaching 99.3% of the performance level of ChatGPT while only requiring 24 hours of finetuning on a single GPU. QLoRA introduces a number of innovations to save memory without sacrificing performance: (a) 4-bit NormalFloat (NF4), a new data type that is information theoretically optimal for normally distributed weights (b) Double Quantization to reduce the average memory footprint by quantizing the quantization constants, and (c) Paged Optimizers to manage memory spikes. We use QLoRA to finetune more than 1,000 models, providing a detailed analysis of instruction following and chatbot performance across 8 instruction datasets, multiple model types (LLaMA, T5), and model scales that would be infeasible to run with regular finetuning (e.g. 33B and 65B parameter models). Our results show that QLoRA finetuning on a small high-quality dataset leads to state-of-the-art results, even when using smaller models than the previous SoTA. We provide a detailed analysis of chatbot performance based on both human and GPT-4 evaluations showing that GPT-4 evaluations are a cheap and reasonable alternative to human evaluation. Furthermore, we find that current chatbot benchmarks are not trustworthy to accurately evaluate the performance levels of chatbots. A lemon-picked analysis demonstrates where **Guanaco** fails compared to ChatGPT. We release all of our models and code, including CUDA kernels for 4-bit training.[2]

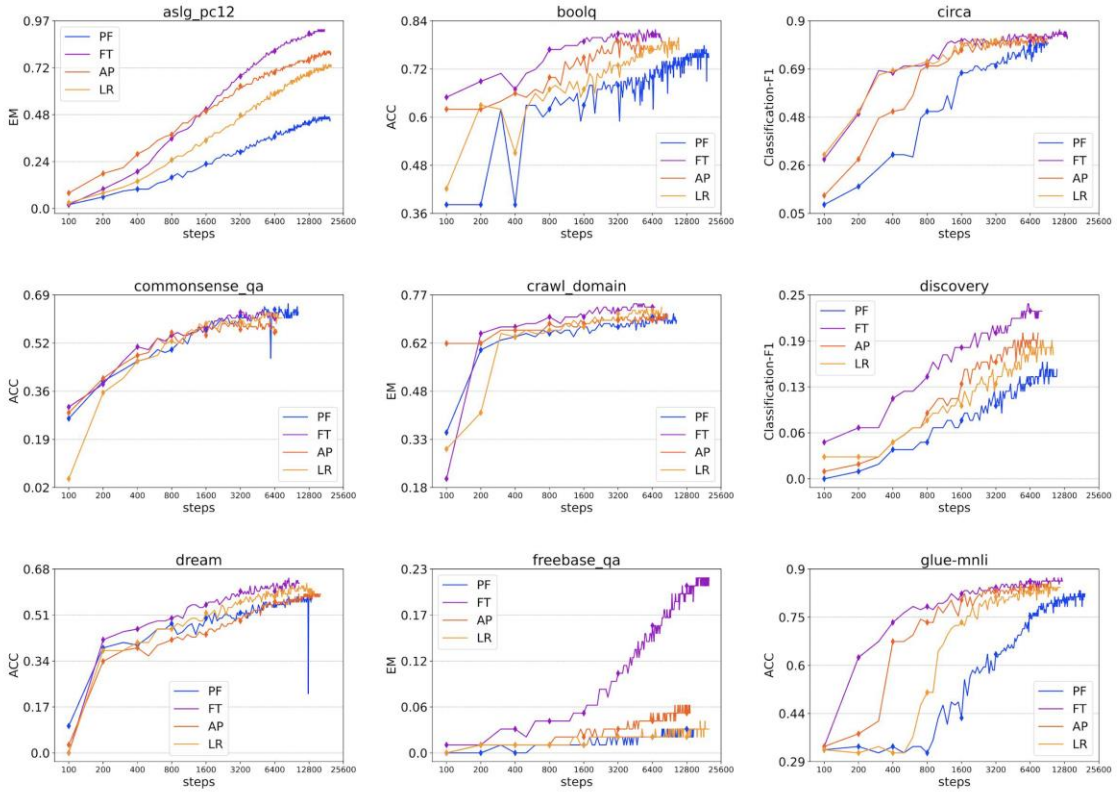


Summary

- Training tends to be more memory-efficient, since we only need to calculate gradients and maintain optimizer state for a small number of parameters.
- These methods are faster to tune than standard full model finetuning.
- It is straight-forward to swap between tasks by swapping in and our just the tuned weights.

Results

If you have the resources, full fine-tuning tends to work the best.



PF: prefix tuning

FT: full fine-tuning

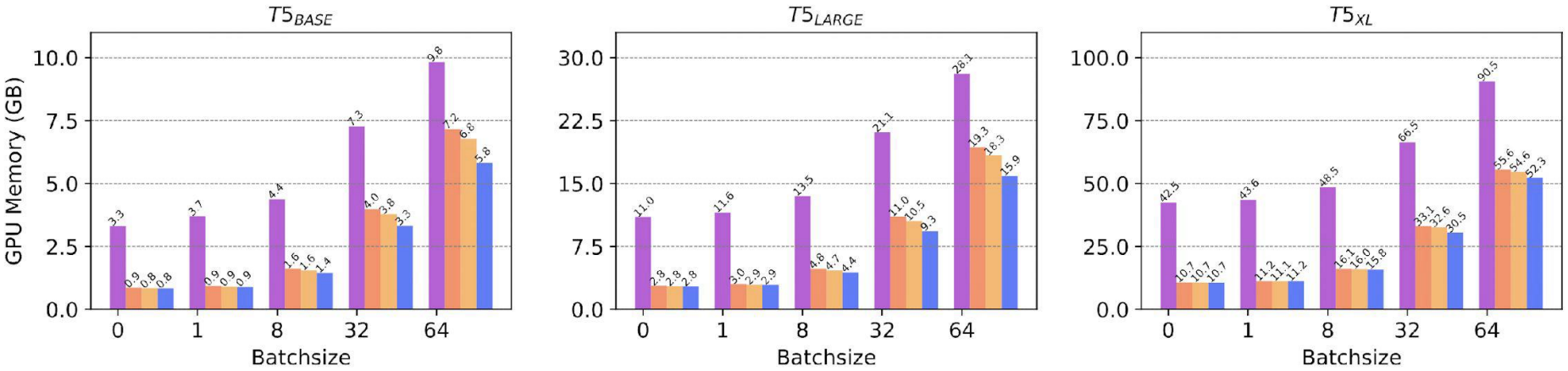
AP: adapter

LR: LoRA

This survey overall found:
Full fine-tuning >
LoRA >
Adapters >
Prefix Tuning >
Prompt Tuning
In terms of performance.

Plots for many more tasks can be found in the paper.

Results



FT: full fine-tuning
AP: adapter
LR: LoRA
BF: BitFit
Prompt tuning and prefix tuning not included because they use the same amount of memory as full fine-tuning

Quiz passcode: Nemo